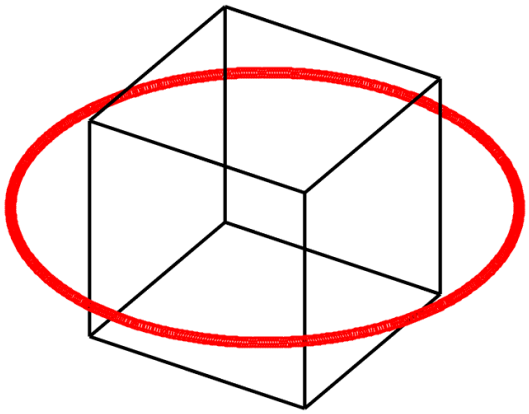
# Edge diffraction toolbox
# EDtoolbox v 0.207
# Manual



Peter Svensson (peter.svensson@ntnu.no)

Acoustics group, Department of Electronic Systems, NTNU

March 22, 2018

# 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 this version,'`EDtoolbox`' v 0.2 (and following numbers, on the format 0.2xx), external scattering problems for convex bodies can be studied. Four different solution approaches are available, each with its own function:

- `EDmain_convexESIE` gives a frequency-domain solution with higher-order diffraction using the edge source integral equation (ESIE). This method gives remarkably accurate results for convex bodies regardless of frequency, except for certain receiver source/positions where singularities limit the accuracy.
- `EDmain_convexESIEBEM` gives a frequency-domain solution with the ESIEBEM approach. The ESIEBEM approach avoids the problems with certain receiver positions, at the price of a bit longer calculations. A set of temporary receivers are distributed over the scattering object surfaces, and the surface sound pressure is computed (using the `EDmain_convexESIE`). In a post-processing step, the sound pressure at the original receivers, so-called field points, is computed by propagating the surface sound pressure with the Helmholtz integral. So-far (v 0.207) rectangular and triangular scattering object faces can be handled, but not general quadrilaterals.
- `EDmain_convexESIEtime` gives a time-domain solution, but so-far (v 0.207) only for first-order diffraction. This is only for special problems.
- `EDmain_convex_time` gives a time-domain solution which computes each diffraction order separately, and with a computation complexity which grows exponentially with diffraction order.

It should be pointed out that polyhedral representations of smooth bodies will give very slow numerical convergence for second- and higher-order diffraction components.

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.

## 1.1 Brief toolbox history

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 all possible geometry cases with a single main program, and that contributed to the complexity. So, in the late fall of 2017 quite a thorough clean-up process was started.

`EDBtoolbox`

In the beginning, a version called `EDBtoolbox` computed impulse responses in interior and exterior geometries, with low-order combinations of specular reflec-

tions and diffractions. This version was developed during the work with Rendell Torres, [3] which in turn built on the first paper with Fred and Vanderkooy, [2]. A major expansion was done in 2003, so that up to six orders of diffraction and specular reflections, and most, but not all, possible combinations could be handled. Also, a limited visibility test of edges was done, using a small number of discrete points along each edge. A handling of the zone boundary singularity for the first-order diffraction was introduced, based on the paper [7]. This version was later called `EDB1toolbox`, to mark that another version, `EDB2toolbox`, included frequency-domain first-order diffraction. The frequency-domain expression was based on the paper [5].

`ESIEtoolbox`

A limitation with the `EDBtoolbox` was that very high orders of diffraction were impossible to handle, because of an exponential growth in computation time with diffraction order. The reformulation of higher-order diffraction into an integral equation made it possible to handle almost arbitrarily high orders of diffraction, because of a linear growth in computation time with diffraction order. The first version, `ESIE0toolbox`, handled only external, convex scattering objects, which was the topic of the paper [4].

A time-domain version of the integral equation formulation, for convex external scattering problems, [6], was implemented in the `ESIE1toolbox`. As a final step, non-convex geometries were permitted in the `ESIE2toolbox`. By then, the program structure was very complex.

`EDtoolbox`

The restructuring of the program is done such that a first version of the main function, `EDmain_convexESIE`, handles only external, convex scattering problems in the frequency domain. It might seem to do nothing more than the `ESIE0toolbox`. However, a large section of code from the `ESIE0toolbox` has been excluded by making specialised functions for first-order specular reflections, since no combinations of specular reflections and diffractions are possible for convex external scattering problems. Furthermore, handling of the first-order diffraction singularity for zone boundaries was finally implemented for the frequency-domain implementation. Another main function, `EDmain_convexESIEBEM`, implemented the ESIEBEM approach in version 0.201. Finally, a first step towards a complete (that is, permitting very high orders of diffraction) time-domain implementation is taken with the function `EDmain_convexESIEtime`. A fourth function, `EDmain_convex_time`, uses the pre-integral-equation approach by computing each diffraction order separately. This function does it quite efficiently because the case of convex scattering bodies requires very simple visibility tests.

## 1.2 License

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

## 1.3 Acknowledgements

Many people have contributed on a smaller and larger scale during more than twenty years of work. The last year, the collaboration with Sara Martin, Jan Slechta and Jason Summers is acknowledged in particular.

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 Overview

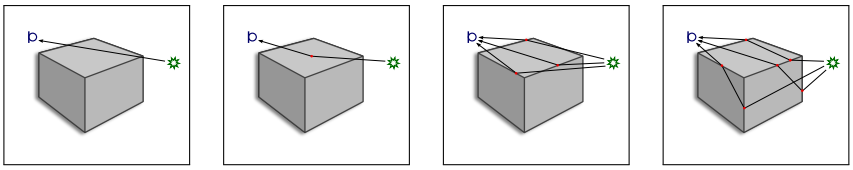
The EDtoolbox computes the scattered, that is, reflected + diffracted, sound pressure for a scattering object which is ensonified by one or more monopole sources. As of version 0.2, geometries can be external and convex only, and frequency-domain formulations have been implemented in addition to some time-domain formulations. The only supported boundary condition is the ideally rigid surface (Neumann boundary condition). The sound pressure at a receiver is decomposed according to this model[1]:

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

where HOD stands for higher-order diffraction. See Fig. 1 for an illustration of some of these terms. It should be noticed that all of these components are subject to visibility tests. For the direct sound component this implies that its value is either exactly the same as the free-field sound pressure, a half, or zero,

$$p_{\text{direct}} = p_{\text{free-field}} \begin{cases} 1, & \text{if the path is unobstructed} \\ 0.5, & \text{if the path passes exactly through an edge} \\ 0, & \text{if the path is obstructed} \end{cases}$$

where "path" refers to the path from the source to the receiver.



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

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

The resulting sound pressure is stored as different variables for the different calculation approaches, as described in Table 1.

[1]With the so-called ESIEBEM approach, this sound field model is used to find the sound pressure at intermediate receiver positions that are automatically distributed along the surface of the scattering object. In a subsequent post-processing stage, it is the total surface sound pressure, $p_{\text{total}}$, which is used in the Helmholtz integral and propagated to the original receiver positions. As a consequence, in this ESIEBEM post-processing stage, the sound field at the original receiver positions is not decomposed into the components of Eq. (1).

Table 1: `EDtoolbox` - Result variables

| **Calculation approach: main function** | **Frequency-domain** | **Time-domain** |
|---|---|---|
| EDmain_convexESIE | `tfdirect, tfgeom, tfdiff, tfinteqdiff` | — — |
| EDmain_convexESIEBEM | `tftot` | — |
| EDmain_convexESIEtime | — — — | `irdirect, irgeom, irdiff, (irinteqdiff)`[1] |
| EDmain_convex_time | — — | `irdirect, irgeom, irdiff, irhod` |

[1] As of version 0.207, the higher-order diffraction has not been implemented in the time-domain version of the ESIE.

Quite different methods are used for the calculation of these components:

- `tfdirect` and `tfgeom` (or `irdirect` and `irgeom`) are the geometrical acoustics (GA) terms[2], and they are computed with explicit expressions, given by the original sources and image sources locations. The visibility factor, very near the zone boundaries, must be determined quite precisely.

- `tfdiff` (or `irdiff`) is the first-order diffraction term, which is computed as a sum of explicit integrals, one for each visible edge. Each such integral is solved to a high accuracy using Matlab's `QUADGK` function, except for a small part of the integration range around near-singularities, where the integral is solved analytically, using the approach in [7], or its corresponding frequency-domain version. In the same way as for the GA terms, the visibility factor near zone boundaries must be determined precisely, and match the visibility of the GA terms.

- `tfinteqdiff` is the sum of second- and higher-order diffraction, up to a chosen diffraction order, which is the same as the iteration order in the Neumann series approach for solving a matrix equation[3]. The matrix equation results from using the Nystrom method to solve the underlying edge source integral equation (ESIE), from [4]. The integral is split up into a sum of integrals, one for each straight edge. For each edge, a Gauss-Legendre (G-L) quadrature scheme is used, with a chosen order specified by the user, through a parameter called `controlparameters.ngauss`, for the longest edge of the scattering object, and proportionally fewer for shorter edges. The error convergence for that solution approach is $O(n_{\text{gauss}}^2)$ (because of the properties at the integration range endpoints). A minimum of three-four such quadrature points per wavelength [4] is re-

[2]Note that in the result file, the variable named `tfgeom` gives only the specular reflection, not the GA solution, which should be the sum of the direct sound and the specular reflection.

[3]To be precise, the iteration order is the diffraction order - 2. As of version 0.207, no automatic iteration truncation has been implemented.

[4]A G-L quadrature scheme does not distribute the points uniformly, but the number of quadrature points for one edge can give an average discretization step size.

quired to keep the maximum error below approximately 1e-2. The solution, given by Eq. (1), consists of three terms that are computed to a very high accuracy, and a fourth term which is computed with less accuracy. The first three terms are contributing to some receiver positions but not to others, which makes the error analysis not very straightforward.

- `irhod` contains higher-order diffraction impulse responses, with each diffraction order computed separately. That is, second-order diffraction is computed as a two-dimensional integral, third-order diffraction as a three-dimensional integral etc. This implies that the computation time increases exponentially with diffraction order. For the frequency-domain implementation, using accurate quadrature schemes, this becomes impossibly time-consuming already for second-order diffraction. On the other hand, time-domain computations are possible to carry out thanks to the Dirac pulse nature of the integrand.

It should be pointed out that 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), and that is why the output variables have those names. 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. As mentioned 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

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

where $U_0$ is the volume velocity of the monopole. 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.

## 2.1 Limitations

The calculations of higher-order diffraction are based on the Edge Source Integral Equation (ESIE) in [4], as mentioned above. Remarkably accurate results have been shown for convex, rigid scattering bodies, for any frequency down towards 0. It is difficult to show why this approach is so accurate, since this approach is not a standard solution method for the Helmholtz equation. It is clear that first-order diffraction is computed by an expression which gives the exact result for an infinite wedge, [5]. The introduction of directional edge sources permits the extension to multiple diffraction, as long as the scattering body is convex, [9], and this seems to yield very accurate results. There are, however, numerical challenges due to a singularity in the edge source directivity function, and this poses numerical challenge in four ways.

### The singularity for first-order diffraction

As illustrated in Fig. 2, the GA terms have discontinuities at the so-called zone boundaries (ZB), and the first-order diffraction term must have corresponding discontinuities. This first-order diffraction discontinuity is caused by a singular integrand, and in [7], this singularity was handled via a series expansion of the integrand, which made an analytical integration possible for the small singular part of the integration range. As mentioned above, this is implemented, for the time- and frequency-domain functions in `EDtoolbox`.

### The singularity in the propagation integral for the ESIE

Whenever a receiver position is close to one of the infinite planes that a finite polygon of the scattering polyhedron belongs to, see Fig. 3 (c), the edge source directivity singularity makes the result converge very slowly, [8]. The so-called ESIEBEM method can overcome this, as shown in [10], and it is implemented in `EDmain_ESIEBEM` for polyhedra with rectangular or triangular faces.

### The singularity in the source term for the ESIE

Equivalently to the receiver position; whenever a source is close to one of the infinite planes that a finite polygon of the scattering polyhedron belongs to, see Fig. 3 (a), the edge source directivity singularity makes the computations numerically challenging. This is not yet handled by the v 0.2 of the `EDtoolbox` but will be attacked in upcoming versions.

### The singularity in the solving of the ESIE

The matrix equation, which is solved via a Neumann series iteration, will have numerical problems if a scattering body has some wedges that are close to 180 degrees, see Fig. 3 (b). This implies that smooth scattering bodies are not efficiently modeled by the ESIE. There is no known solution yet for improving that situation.

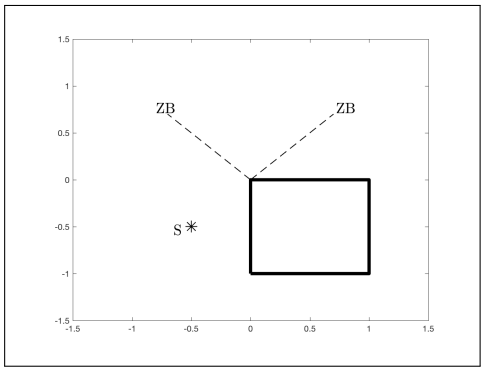
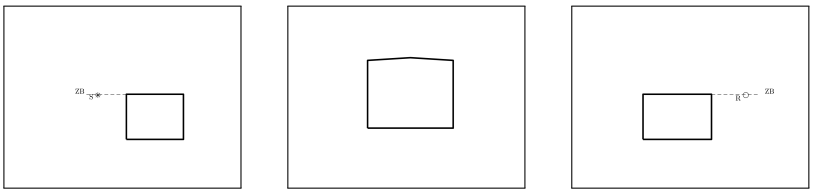



Figure 2: Illustration of the zone boundaries, ZB, for one of the edges of a scattering object, which cause discontinuities for the GA terms, and the first-order diffraction.



(a) (b) (c)

Figure 3: Illustration of situations where the singularities for the higher-order diffraction cause numerical problems, for the (a) source term, $\mathbf{q_0}$ computation, (b) iteration for the determining $\mathbf{q}$, (c) propagation of $\mathbf{q}$ to $p$ at the receiver.

## 2.2 Implemented versions

As of version 0.2 of the toolbox, the cases described in Table 2 have been implemented.

Table 2: `EDtoolbox` - Implemented cases

| **Geometrical domain** | **Frequency-domain** | **Time-domain** |
|---|---|---|
| External, convex scattering using ESIE | `EDmain_convexESIE` | `EDmain_convexESIEtime` (No HOD as of v0.207) |
| External, convex scattering using ESIEBEM | `EDmain_convexESIEBEM` | Not implemented |
| External, convex scattering, solving each diffraction order separately | Not implemented (Computing multidimensional FD integrals is prohibitively expensive) | `EDmain_convex_time` |
| Internal problems, or external, non-convex | Not implemented yet | Not implemented yet |

## 2.3 Needed functions from Matlab Central

Two functions developed by others are required for `EDtoolbox`, and they must be downloaded from Matlab Central, and 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 program, the directory with result files is scanned for existing files with the same hash value, which makes it possible to use that existing file instead of calculating a new.

- `lgwt`, developed by Greg von Winckel. This function gives the Gauss-Legendre nodes and weights.

## 2.4 How to run the `EDmain_xxx` functions

The various `EDmain_xxx` functions are all run by assigning values to six input structs,

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

each with a number of fields, that are further described in Section 4.

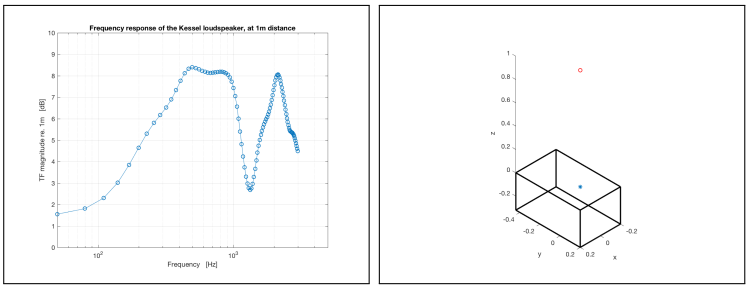
Attempts have been made to give many default values to settings, such that it can be easy to get started. As one example, the script below defines a cuboid loudspeaker box, a point source right at the box, and a receiver 1 m away. The function `EDmain_convexESIE` is run, and after the calculations have been run, the result files are loaded and a frequency response is plotted. The example is taken from the collection of scripts in `EDexamples`. The script `EDexample_LspKessel_minimal.m` has some additional explaining text, but gives the same results as the script below. Fig. 4 shows the resulting output and the model with the source and receiver locations indicated.

The response in Fig. 4 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, 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 will indeed come very close to $1/r$.

The response is computed up to 3000 Hz. For higher frequencies, a larger number of edge points (setting `controlparameters.ngauss`) would be needed. One can estimate that 3 edge points per shortest wavelength are needed.

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

%----------------------------------------------------------------------
% Define the scattering object = the "Kessel" loudspeaker box

corners = [ -0.20 -0.44 -0.32;  0.20 -0.44 -0.32;  0.20 0.20 -0.32; ...
 -0.20 0.20 -0.32;  -0.20 -0.44 0;  0.20 -0.44 0;  0.20 0.20 0;  -0.20 0.20 0];
planecorners = [ 1 4 3 2;  5 6 7 8;  1 2 6 5;  3 4 8 7;  2 3 7 6;  1 5 8 4];

%----------------------------------------------------------------------
% Give calculation parameter values, including S and R positions

geoinputdata = struct('corners',corners,'planecorners',planecorners);
Sinputdata = struct('coordinates',[0 0 0.00001]);
Rinputdata = struct('coordinates',[0 0 1]);
controlparameters = struct('frequencies',linspace(50,3000,100));
filehandlingparameters = struct('filestem',filestem)
```

```
filehandlingparameters.outputdirectory = [infilepath,filesep,'results'];

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

EDmain_convexESIE(geoinputdata,Sinputdata,Rinputdata,struct,controlparameters, filehandlingparameters);

%------------------------------------------------------------------------
% Load and present the results, and plot the geometry model

eval(['load ',filehandlingparameters.outputdirectory,filesep,...
filehandlingparameters.filestem,'_tfinteq.mat'])
eval(['load ',filehandlingparameters.outputdirectory,filesep,...
filehandlingparameters.filestem,'_tf.mat'])
tftot = tfdirect + tfgeom + tfdiff + tfinteqdiff;

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

eddatafile = [filehandlingparameters.outputdirectory,filesep,...
filehandlingparameters.filestem,'_eddata.mat',3];

EDplotmodel(eddatafile,1)
```




(a) (b)

Figure 4: The "Kessel" loudspeaker example. (a) The frequency response at 1m distance, for a point source, computed with `EDmain_convexESIE`. (b) The loudspeaker model, with source and receiver positions.

## 2.5 Known bugs

A few bugs are known, as of v 0.207. Two bugs are of the same character:

- If the receiver is hidden behind the scattering object, but the direct sound path happens to pass exactly through two edges, then the path is not detected as obstructed. This failure is revealed by the test function `EDverify`, test case number 8.
- Similarly, if the direct sound path passes exactly through two corners, the path is not detected as obstructed. This failure is revealed by the test function `EDverify`, test case number 7.

Another bug, or unclarity of the underlying equations, is that if the direct sound passes exactly through one corner, its amplitude is presently computed

as 0.5*0.5 = 0.75 times the unobstructed sound wave amplitude. This does, however, not give a continuous sound pressure amplitude through that point.

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

## 2.6 Planned developments

- The output data should contain markers for when a singularity was encountered.
- A time-domain version of the ESIE will be implemented.
- An automatic iteration stop criterion, for the integral equation solving, could quite easily be implemented. It would be based on some norm of the iterated vector edge source signals.
- An automatic edge point refining could be implemented which would run a few different numbers of edge points, while estimating the relative error.

# 3 Geometry format

The `EDtoolbox` handles only polyhedra, including polygonally shaped thin discs/ plates. In the `EDtoolbox`, a polyhedron is defined in terms of 'corners' (vertices) and 'planes' (faces/polygons). These can either be specified directly in the input struct geoinputdata (fields `.corners` and `.planecorners`, as in the example in section 2.4), or in a separate file of the .cad-format, which is a format exported by the CATT-Acoustic software [1]. Fig. 5 shows a simple example: a cuboid box.

## 3.1 Corners

The `.corners` field is straightforward: it is a matrix of size [ncorners,3] where row n contains the x-,y- and z-coordinates of corner number n. If the .cad-file had a non-contiguous numbering of the corners, a renumbering will be done for the EDtoolbox, starting with number 1. For the example in Fig. 5, 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; ...
```

## 3.2 Planes

The `.planecorners` field is a matrix of size [nplanes,nmaxnumberofcornersperplane] where row n gives the corners that define plane n.
The example in Fig. 5 would have its planes defined as

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

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, you must specify both sides of the plane!
- Some geometry generating software splits up polygons into triangles, but `EDtoolbox` can not handle co-planar triangles! You must construct as large polygons as possible for each face of the polyhedron!

## 3.3 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. 5 (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
```

## 3.4 Excluding some edges

There are some limited possibilities to create geometrical models with just a few edges, using the parameter `geoinputdata.firstcornertoskip`. By giving a corner number of the model to that parameter, all edges with at least one corner which has a number like that parameter value, or higher, will be deactivated. Another possibility is to use a cad-file and give a plane the type `TOTABS`. Then, all edges that are connected to that plane will be deactivated. Using these techniques, it is possible to simulate, e.g., just the top edges of a noise barrier.

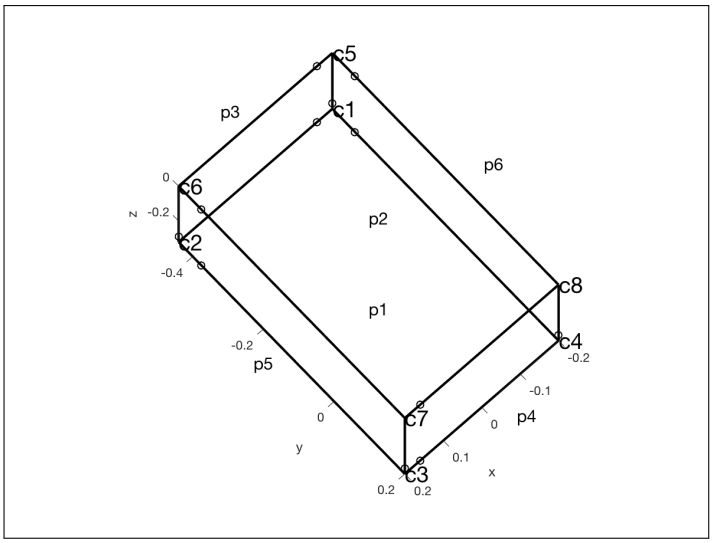



Figure 5: Illustration of a cuboid scattering object. Corner numbers and plane numbers are indicated.

# 4 Input data

The main functions, `EDmain_xxx`, are run with six structs containing all input parameters:

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

These six structs are described in Tables 3 - 9.

Table 3: Input data struct `geoinputdata`

| Field name | Required? | Default value | Size |
|---|---|---|---|
| `.geoinputfile` | Alt. A (see below) | — | — |
| `.corners` | Alt. B (see below) | — | [ncorners,3] |
| `.planecorners` | Alt. B (see below) | — | [nplanes,nmax][1] |
| `.firstcornertoskip` | — | 1e6 [2] | |

Three alternatives exist for 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.
Priority will be given to the `.geoinputfile` if both alternatives A and B are given.
See section 3 for more information on the geometry format.
[1] The value nmax is the maximum number of corners per plane. Note that if one plane has many more corners than others, the rows for those other planes will (have to) contain many zeros, after the few corner values.
[2] The field `.firstcornertoskip` implies that all edges with at least one corner number having the value of `.firstcornertoskip`, or higher, will be deactivated. This gives the possibility to study cases with a subset of all the edges of a model, see section 3.4.

Table 4: Input data struct `Sinputdata`

| Field name | Required? | Default value | Size/value |
|---|---|---|---|
| `.coordinates` | Yes | — | [nsources,3] |
| `.doaddsources` | — | 0 | 0 or 1[1] |
| `.sourceamplitudes` | — | 1 | [nsources,nfreq] |
| `.doallSRcombinations` | — | 1 | 0 or 1[2] |

[1] If this value is set to 1, the contributions from all sources will be added and saved in a single transfer function, after being multiplied by the values in the vector `.sourceamplitudes`. This is a straightforward way to simulate extended sources, or vibration patterns. See section 5 for a description of the scale values.
[2] 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 `doallSRcombinations` to 0. This is relevant, e.g., for computing backscatter cases, with the receiver in the same location as the source, but do this for a number of incidence angles.

Table 5: Input data struct `Rinputdata`

| Field name | Required? | Default value | Size |
|---|---|---|---|
| `.coordinates` | Yes | — | [nreceivers,3] |

Table 6: Input data struct `envdata`

| Field name | Required? | Default value | Size |
|---|---|---|---|
| `.cair` | — | 344 | — |
| `.rhoair` | — | 1.21 | — |

Table 7: Input data struct `controlparameters`

**O** = obligatory, d(value) = gets default value if not specified,
— = ignored and removed

| | **EDmain_convex function** | | | |
|---|---|---|---|---|
| **Field name** | **ESIE** | **ESIEBEM** | **ESIE_time** | **_time** |
| `.directsound` | d(1) | d(1) | d(1) | d(1) |
| `.skipfirstorder`[1] | d(0) | d(0) | d(0) | d(0) |
| `.Rstart`[2] | d(0) | d(0) | d(0) | d(0) |
| `.difforder` | d(15) | d(15) | d(1000)[3] | **O**[4] |
| `.docalctf`[5] | d(1) | d(1) | — | — |
| `.frequencies` | **O** | **O** | — | — |
| `.discretizationtype`[6] | d(2) | d(2) | d(2) | — |
| `.ngauss`[7] | d(16) | d(16) | d(16) | — |
| `.surfacegaussorder`[8] | — | d(5) | — | — |
| `.docalcir`[5] | — | — | d(1) | d(1) |
| `.fs` | — | — | d(44100) | d(44100) |
| `.savealldifforders` | — | — | — | d(0) |

[1] If this parameter is set to 1, the direct sound, specular reflection, and first-order diffraction will not be computed. Please note that resultfiles with the extension `_ir` or `_tf` will still be saved, with empty variables.
[2] The parameter `.Rstart` determines the phase of the final transfer function (or the definition of time zero in time-domain calculation alternatives). To simulate an incoming plane wave with amplitude 1, and phase zero, at the origo, then `.Rstart` should be set to the distance to the far-away point source.
[3] For `EDmain_convexESIE_time`, the `difforder` is actually the number of time-marching step, which is not the same as diffraction order: one diffraction order is spread out over many time steps!
[4] For `EDmain_convex_time`, `difforder` can not be higher than 6.
[5] If the field `.docalctf` (or `.docalcir`) is set to 0, edges will be derived and source/receiver visibility will be computed. Please note that to have any use for these calculations, you must specify in the struct `filehandlingparameters` that the proper geometry information is saved.
[2] The sampling frequency, `fs`, and the parameters `.docalcir` and `.HODcalculation`, are used in upcoming time-domain calculation functions, but are not read/used by EDmain_convexESIE.
[3] A list of frequencies must be specified for the main function EDmain_convexESIE, and other upcoming frequency-domain versions (unless `.docalctf` is 0). It is not needed for time-domain versions.
[5] This parameter specifies how many points along each edge will be tested for visibility. This is irrelevant for convex scattering bodies since either the whole edge or no part of an edge is visible. It is relevant for upcoming calculation alternatives for non-convex geometries.
[6] The value 0 implies a uniform discretization of the edges. The value 2 gives a Gauss-Legendre discretization. The value 1 is obsolete/not used.
[7] The value `.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 two. [8] The value `.gauss` 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 two.

Table 8: Input data struct `filehandlingparameters`

| Field name | Required? | Default value |
|---|---|---|
| `.outputdirectory` | Yes [1] | Same as geoinputfile [2] |
| `.filestem` | Yes[1] | Name of cad-file |
| `.suppressresultrecycling` | — | 0 |
| `.savecadgeofile` | — | 0 |
| `.saveSRdatafiles` | — | 1 |
| `.saveeddatafile` | — | 1 |
| `.saveed2datafile` | — | 1 |
| `.savesubmatrixdata` | — | 1 |
| `.saveinteqsousigs` | — | 0 |
| `.loadinteqsousigs` | — | 0 |
| `.savepathsfile` | — | 1 |
| `.saveISEStree`[3] | — | 0 |
| `.savelogfile` | — | 1 |
| `.savediff2result` | — | 0 |

[1] If the geometry is given in the form of the input fields `.corners` and `.planecorners`, then the fields `.outputdirectory` and `.filestem` must be specified.

[2] Note that as default, a folder called "results" will be generated in the directory of the geoinputfile (if it doesn't already exist). So, `outputdirectory` will be '<same folder as geoinputfile>/results'. All result files will be saved in that results folder.

[3] This file is not used by `EDmain_convexESIE` or `EDmain_convexESIE_ir` but will be used in future versions for internal problems, and external, non-convex problems.

# 5 Output data

## 5.1 The resulting sound pressure/transfer functions

The main functions, `EDmain_xxx`, can generate several types of optional output files, as indicated by table 8. There are, however, some results files that will always be generated, with the most important output: the resulting sound pressure in the form of transfer functions or impulse responses. Below, they are presented for each `EDmain_xxx` function.

### 5.1.1 `EDmain_convexESIE`

Two result files will contain transfer functions that correspond to the four terms in Eq. (1)

`<filestem>_tf.mat` (with `tfdirect, tfgeom, tfdiff`)

`<filestem>_tfinteq.mat` (with `tfinteqdiff`)

You will have to add these terms in your own scripts, after loading the two files above, as can be seen in the example script in section 2.4:

```
tftot = tfdirect + tfgeom + tfdiff + tfinteqdiff
```

In some cases, one might not be interested in the direct sound, and then

```
tfreflection = tfgeom + tfdiff + tfinteqdiff
```

Each of these transfer functions will have the same size:

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

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`.

### 5.1.2 `EDmain_convexESIEBEM`

As specified in Table 1, and discussed in the same section, the ESIEBEM approach does not decompose the sound pressure, but rather gives the single total sound pressure:

`<filestem>_tfesiebem.mat` (with `tftot`)

The transfer function will have the size:

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

### 5.1.3 EDmain_convexESIE_time

The two output files will be

<filestem>_ir.mat (with irdirect, irgeom, irdiff)

<filestem>_irinteq.mat (with irinteqdiff)

All that was described in Section 5.1.1 applies here too, with irxxxx instead of tfxxxx. However, the size of the resulting matrices has a size nirsamples along the first dimension, rather than nfrequencies.

### 5.1.4 EDmain_convex_time

The two output files will be

<filestem>_ir.mat (with irdirect, irgeom, irdiff)

<filestem>_irhod.mat (with irhod)

Here, the variable irhod will contain 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 size [nfreq,nR,nS] (if doaddsources = 0) or [nfreq,nR,1] (if doaddsources = 1, or nsources = 1).

## 5.2 Timing data

The file named <filestem>_tfinteq.mat will contain a variable called timingstruct. This struct contains timing data for the different parts of EDmain_convexESIE as shown in Table 9.

One example of values is given below for the example EDexample_LspKessel_minimal in Section 2.4.

```
timingstruct =
struct with fields:
geoinput:  0.0092
 edgedata:  0.0248
 Sdata:  0.0099
 Rdata:  0.0050
 findpaths:  0.0264
 maketfs:  [0.4092 0.0046 0.0014 0.4026]
 edgetoedgedata:  0.0246
 submatrixdata:  0.0137
 integralequation:  [53.6891 0.0904 0.0158 0.4994 0.0140]
```

Table 9: The `timingstruct`

| Field name | Function which is timed | One or several values |
|---|---|---|
| `.geoinput` | `EDreadcad`, or `EDreadgeomatrices` | 1 |
| `.edgedata` | `EDedgeo` | 1 |
| `.Sdata` | `EDSorRgeo` | 1 |
| `.Rdata` | `EDSorRgeo` | 1 |
| `.findpaths` | `EDfindconvexGApaths` | 1 |
| `.maketfs` | `EDmakefirstordertfs` | [1,4][1] |
| `.edgetoedgedata` | `EDed2geo` | 1 |
| `.submatrixdata` | `EDinteg_submatrixstructure` | 1 |
| `.integralequation` | `EDintegralequation_convex_tf` | [1,5][2] |

[1] The four values are times for:
(1) the entire function call,
(2) generating direct sound component(s),
(3) generating specular reflection(s),
(4) generating first-order diffraction

[2] The five values are times for:
(1) the entire function call (all frequencies),
(2) generating the **H**-matrix for one frequency (done only if `difforder`$> 2$),
(3) compute the source term **q**_0 for one frequency,
(4) compute **q** via iterations, for one frequency,
(5) propagate the edge source signals **q** to the receiver point, for one frequency

## 5.3 Other output data

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

- `EDsettings`, which is a cell variable that contains all 6 input structs, and the EDtoolbox version number:
  - `EDsettings{1} = geoinputdata`
  - `EDsettings{2} = Sinputdata`
  - `EDsettings{3} = Rinputdata`
  - `EDsettings{4} = envdata`
  - `EDsettings{5} = controlparameters`
  - `EDsettings{6} = filehandlingparameters`
  - `EDsettings{7} = EDversionnumber`
- `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.

## 5.4 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.

```
######################################################################
# EDmain_convexESIE, v.  0.108 (2Feb2018)
# filestem for results:  EDexample_LspKessel_minimal

 EDreadgeomatrices (8 corners and 6 planes), time:  0.009236 s
 EDedgeo, (12 edges), time:  0.024809 s
 EDSorRgeo(S), (1 source(s)), time:  0.009944 s
 EDSorRgeo(R), (1 receiver(s)), time:  0.005021 s
 EDfindconvexGApaths (100 frequencies)
                        Total time:  0.02637 s
 EDmakefirstordertfs (100 frequencies)
                        Total time:  0.40925 s.  Parts, for all frequencies, as below
                        Generate the direct sound:  0.004572 s
                        Generate the specular reflections:  0.001432 s
                        Generate the first-order diffraction:  0.40264 s
 EDed2geo, time:  0.024557 s
 EDinteg_submatrixstructure, (252 submatrices, out of 432, to compute), time:  0.013727 s
                        (Edges discretized with:  8 to 16 discretization points)
                        (Avg.  "edge element" size:  0.04 m.
                        OK up to 2867 Hz (3 discret.  points per wavelength))
                        (9248 edge source signals to compute)
                        (628864 non-zero elements in the IE matrix)
 EDintegralequation_convex_tf (100 frequencies.  Diffraction order:  15)
                        Total time:  53.6891 s.  Parts, for one freq, as below)
                        Compute the H-matrix:  0.090421 s
                        Compute Q_firstterm:  0.015794 s
                        Compute Qfinal:  0.49939 s
                        Compute the result at the receiver(s):  0.014038 s
```

## 5.5 Other output quantities

The contents in the optional output files are described in table 10, and some of the structs are further detailed in tables 13 - 18 in the Appendix, section 10.

Table 10: Contents of the optional output files

| **Output file**[1] | **Contents**[2] |
|---|---|
| _cadgeo | planedata, extraCATTdata |
| _eddata | planedata, edgedata, EDinputdatahash[3] |
| _Sdata, _Rdata | Sdata or Rdata, EDinputdatahash[3] |
| _ed2data | planedata, edgedata, edgetoedgedata, EDinputdatahash[3] |
| _paths | firstorderpathdata, EDinputdatahash[3] |
| _submatrixdata | Hsubmatrixdata, EDinputdatahash[3] |

[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 marked.
[3] The text string in `EDinputdatahash` will be determined from different subsets of all settings for each output file.

# 6 Results: convergence, accuracy, computational cost, limitations

## 6.1 Convergence rate

A described in section 2, for the computation of higher-order diffraction, an integral equation is solved by discretizing the edges. The integral is a Fredholm equation of the second kind, and the Nystrom method is used to solve it. This method implies a discretization of the edges, preferrably following a Gauss-Legendre (GL) quadrature rule, [12]. The GL rule would give an exponential convergence rate for increasing quadrature order (called "number of edge sources" here), if the integrand is smooth and with finite derivatives everywhere. For the ESIE, it is not possible to achieve this exponential convergence due to endpoint singularities, so the error has been found to follow $O(n^{-2})$, where $n$ is the number of edge sources per edge.

The parameter `controlparameters.ngauss`, with default value 16, determines the quadrature order, or number of discretization points, or "edge points", per longest edge of the model, and with proportionally smaller numbers for the shorter edges, with a minimum number of two discretization points per edge. In the same way as for the boundary element method and other numerical methods based on discretizing space, a number of such discretization points are required per wavelength. In addition, the result is converging to some finite result for infinitely many discretization points, and hopefully this final result is the same as the true solution. These aspects will be demonstrated here via the example script `EDexample_Cube_convergenceDC`. As a first step, the convergence rate will be studied for plane wave incidence of the frequency 0, onto a cube. For receivers immediately at the surface, the result should be exactly 1, that is, exactly the same value as in the incident free-field plane-wave.

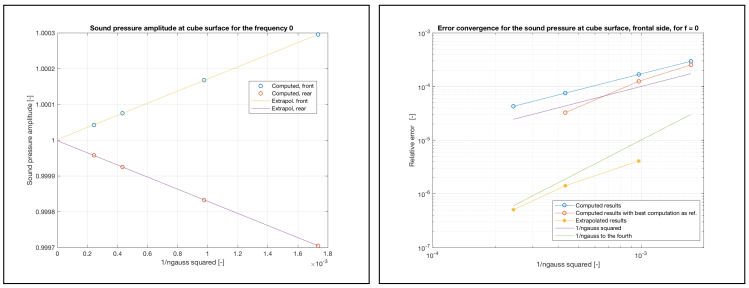
This example demonstrates how to construct a script which runs through a number of values of some setting parameter via a for-loop. Specifically, the example will run a number of edge points (ngauss): 24, 32, 48, and 64 per edge, and furthermore use extrapolation to get a better estimate of the final value. The case is for plane wave incidence (actually a point source at a distance of 1e9 m), and receivers immediately at the surface of a cube. Two receiver points are set, one on the illuminated side of the cube, and the other on the shadow side.

By inspecting the results for the frequency 0, the two receiver points (columns 2 and 3 below), and the four discretizations (see the values in column 1), we see that the results seem to approach 1 as the number of edge points is increased. (`format long` was used to display so many decimals, but we removed the zeros for the integers in the first column in the editing process of this text)

```
>> [ngvec squeeze(allres(1,1:2,:)).']
ans =
24 1.000295120031160 0.999704879966883
32 1.000167794185908 0.999832205812937
48 1.000075361604386 0.999924638395042
64 1.000042610860548 0.999957389139086
```

The next step is to apply extrapolation, to illustrate that the error goes as $O(n^{-2})$, where $n$ is the number of discretization points. Thus, the code below generates the result in Fig. 15 (a).

```
xvec = 1./ngvec./ngvec;
pfront = polyfit(xvec,squeeze(allres(1,1,:)),1);
prear = polyfit(xvec,squeeze(allres(1,2,:)),1);
xvecplot = [0 xvec(1)].';
yvecfrontplot = polyval(pfront,xvecplot);
yvecrearplot = polyval(prear,xvecplot);
plot(xvec,squeeze(allres(1,1:2,:)),'o',xvecplot,yvecfrontplot, xvecplot,yvecrearplot)
xlabel('1/ngauss squared [-]')
ylabel('Sound pressure amplitude [-]')
grid
h = legend('Computed, front','Computed, rear', 'Extrapol.  front','Extrapol, rear');
set(h,'Location','best')
title('Sound pressure amplitude at cube surface for the frequency 0 ')
```




(a) (b)

Figure 6: (a) The sound pressure amplitude at the frontal and rear surfaces of a cube, for plane wave incidence of the frequency 0, as function of 1/(the number of edge points squared). Computed results are for 24,32,48, and 64 points per edge. Linear extrapolations are also shown. (b) The error convergence for the computed results in (a). Extrapolated results are from pairs of computed values.

We can also inspect the relative error for the computed results and the extrapolated results, since we know the exact result. The fourth and fifth columns below are the relative errors for the frontal side and rear side results. The last row is for the extrapolated results, and apparantly, the extrapolation reached an error around 1.8e-6 from our results, that at the best gave us an error of 4.26e-5 (second last row). In fact, if we had used just the two best computations, with 48 and 64 edge points per edge, for the linear extrapolation, we would reach an error of 5e-7.

```
A = [ngvec squeeze(allres(1,1:2,:)).'];
A = [A abs(A(:,2:3)-1)];
A = [A;0 pfront(2) prear(2) abs(pfront(2)-1) abs(prear(2)-1)]
A =
24 1.000295120031160 0.999704879966883 0.000295120031160 0.000295120033117
32 1.000167794185908 0.999832205812937 0.000167794185908 0.000167794187063
48 1.000075361604386 0.999924638395042 0.000075361604386 0.000075361604958
64 1.000042610860548 0.999957389139086 0.000042610860548 0.000042610860914
 0 1.000001811176579 0.999998188823313 0.000001811176579 0.000001811176687
```

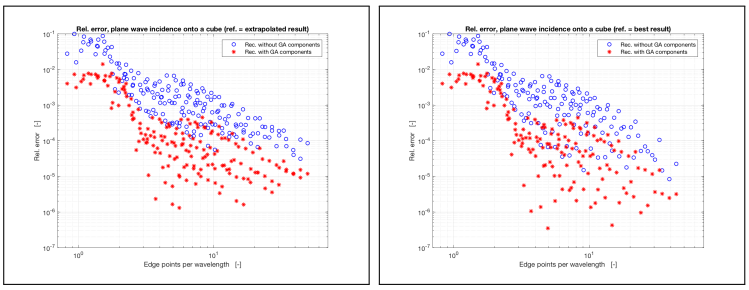
Finally, the error convergence can then be plotted as in Fig. 15 (b), using the code below. The extrapolated values come from each available (consecutive) pair of computed points, so there are three extrapolated values from the four

computed data points. Apparantly, the extrapolated results have a faster convergence rate than the computed data points. The extrapolated results reach the theoretically expected rate, $O(n^{-4})$, for the best computed results, since the coarsest discretizations don't give as linear a relationship for the computed data points; an order two polynomial could have been needed to describe the error convergence better. One extra curve is shown in Fig. 15 (b), and that is the apparent error convergence that results if one chooses the best computed data point as reference (and this might be the only alternative one has, if no other reference results are available). Two observations can be made: firstly, the apparant error is smaller than when the correct reference result is used. Secondly, the rate is not easy to identify because of the non-straight convergence curve.

```
extrapolres = zeros(ncases,1);
for ii = 2:ncases
pfront = polyfit(xvec(ii-1:ii),squeeze(allres(1,1,ii-1:ii)),1);
extrapolres(ii) = pfront(2);
end
relerr = abs(squeeze(allres(1,1,:)).'-1);
relerrcompref = abs( squeeze( allres(1,1,:)).'  -allres(1,1,4));
relerrextrapol = abs(extrapolres-1);

figure
loglog(xvec,relerr,'-o',xvec,relerrcompref,'-o',xvec(2:end),relerrextrapol(2:end),'-*',...
xvec,0.1*xvec, xvec,10*xvec.*xvec)
grid
xlim([1e-4 2e-3])
xlabel('1/ngauss squared [-]')
ylabel('Relative error [-]')
h = legend('Computed results','Computed results with best computation as ref.',...
'Extrapolated results', '1/ngauss squared','1/ngauss to the fourth');
set(h,'Location','best')
title('Error convergence for the sound pressure at cube surface for f = 0')
```

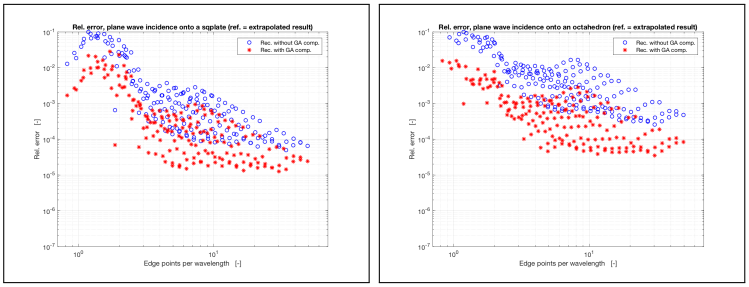
## 6.2 Number of edge points per wavelength

Now, the example `EDexample_Cube_edgepointsperwavelength.m` is used to demonstrate how the number of edge points (ngauss) per wavelength affects the error. This is done by repeating the previous example for a range of frequencies and generate relative errors like in Fig. 15 (b). Both the "best" results (i.e., results computed with the largest number of edge/gauss points) and extrapolated results (from the two best results) will be used as reference results. Results were computed for 20 frequencies, logarithmically distributed between 500 Hz and 5000 Hz, which corresponds to a $kL$-range of 9.1 to 91, where $L$ is the edge length of the cube. Nine discretizations were run, with 12, 14, 16, 20, 24, 32, 48, 64, and 72 points per edge. For each of these frequencies, a reference result estimate was either the best result (with 72 points per edge) or a linear extrapolation as in section 6.1, computed from the two best results (64 and 72 edge points). Then, the relative error was computed for all 180 points, as function of elements per wavelength, and presented in Fig. 16. By inspecting Fig. 16 (a) and (b), one can see that the choice of reference result makes a difference for the smallest errors, but has very little influence for the largest errors. This corresponds to the observation that in Fig. 15 (b), the two curves "Computed results" and "Computed results with best computation as ref." are very close for the large error values (the right end of the curves), while they differ more for small error values (the left end of the curves).




(a) (b)

Figure 7: The relative error for several different frequencies, and edge discretizations (gauss quadrature order), for plane wave incidence onto a cube. Results for the receivers with, and without, a direct sound component, are plotted separately. For each of 20 frequencies, the reference result was, (a) the best data point (highest number of edge points), and (b) extrapolated results from two best computed data points, respectively.



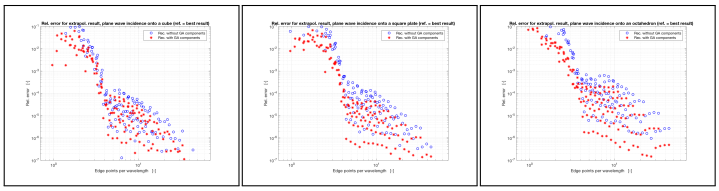

(a) (b)

Figure 8: The relative error for several different frequencies, and edge discretizations (gauss quadrature order), for plane wave incidence onto a (a) square plate, and (b) an octahedron, respectively. For each of 20 frequencies, the reference result was the extrapolated result from the two best computed data points.

Results were also collected for a square plate, using the script `EDexample_Cube_edgepointsperwavelength.m` and an octahedron, with the script `EDexample_Octahedron_edgepointsperwavelength.m`. The same settings were used as for the cube, and the results are presented in Fig. 8, with extrapolated results as reference. Again, it can be seen that whenever the GA component are non-zero, the error is substantially lower. Furthermore, there is a kind of transition around three edge points per wavelength, above which the error is generally below 1e-2. For the octahedron, however, the error is above 1e-2 for a number of cases.

As a last analysis of these results, extrapolated results from the two last calculations, will be used with the "best extrapolated result" as reference. The three scripts `EDexample_Cube_edgepointsperwavelength_extrapol.m`, `EDexample_Squareplate_edgepointsperwavelength_extrapol.m`, and

`EDexample_Octahedron_edgepointsperwavelength_extrapol.m` generate the diagrams in Figure 9. So, instead of using a single computation as a data point, two consecutive computations were used to generate a linearly extrapolated result. These extrapolated results gave errors as shown in Figure 9. Here one can observe that the extrapolated results have generally significantly lower errors than a single computation. The difference in number of edge points for two consecutive edge discretizations was 13% (given by distributing 20 frequencies over a factor 10 in frequency).



(a) (b) (c)

Figure 9: The relative error for extrapolated results, for several different frequencies, and edge discretizations (gauss quadrature order), for plane wave incidence onto a (a) cube, (b) square plate, and (c) an octahedron, respectively. The errors are computed for extrapolated results, that were computed from two different numbers of edge points, differing by around 13%.

We could summarize these findings as:

- A minimum of three edge points per wavelength is enough to reach errors below 1e-2 most of the time.
- Whenever the direct sound and/or specular reflection is non-zero, the error is usually substantially lower. However, a non-zero direct sound and/or specular reflection can also give strong negative interference effects which might lead to larger relative errors.
- By carrying out two computations, with, say, 3.5 and 4 edge points per wavelength, or better, and using extrapolation, the error will be substantially below 1e-3.

The last point above indicates possibilities for an adaptive solution approach where consecutive computations are carried out automatically, using extrapolation for estimating the error, and thereby get an estimate of the current error, and finally use the last extrapolated result for an ultimate result (with unknown accuracy).

## 6.3 Computational cost

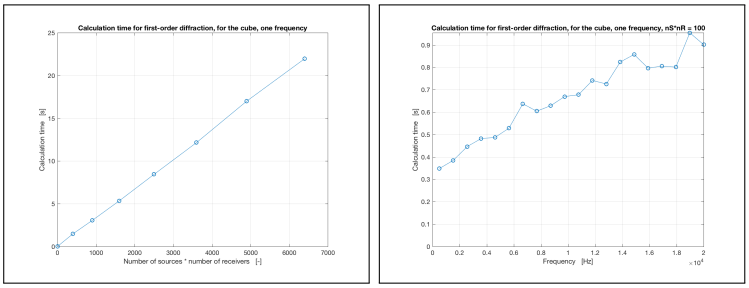
The computational cost can be analyzed using the values that are stored in `timingdata` in the `_tf` and `_tfinteq` files.

### 6.3.1 First-order diffraction

The script `EDexample_Cube_diff1cost.m` runs first-order diffraction calculations for one frequency and a number of sources and receivers. Fig. 10 (a) shows that there is a direct linear dependence on number of sources times number of receivers. The script `EDexample_Cube_diff1costfreq.m` has been prepared to study the influence of the frequency on the first-order diffraction calculation time, and Fig. 10 (b) illustrates that the numerical integration follows an almost linear relationship plus a constant. For this particular example, frequency range, and computer, the relationship is

$$t_{\text{diff.1,cube}} \approx (3,8 + 0,25 f_{\text{kHz}})\, n_S n_R \text{ ms}$$

The numerical constants of this expression are interesting mostly for a comparison with the cost for higher-order diffraction. The values depend on the number of edges that are computed, the length of the edges, and the computer used.




(a) (b)

Figure 10: Calculation time for the first-order diffraction components of the cube, as function of (a) number of sources times number of receivers, for one frequency, (b) frequency, for 100 source and receiver combinations, one frequency.

### 6.3.2 Higher-order diffraction

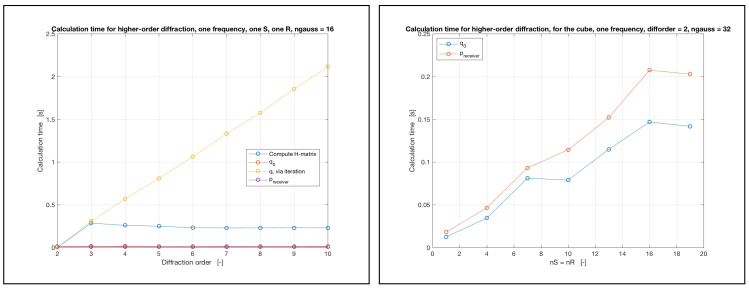
For the higher-order diffraction computations, more factors are affecting the computational cost: in addition to number of sources and receivers, also the diffraction order, and number of edge/gauss points per edge. The script `EDexample_cube_HODcost_difforder` generates the diagram in Fig. 11 (a). The iterative solution part has a linear dependence on diffraction order, from order 3, which implies a constant cost per diffraction order, as expected. In addition, setting up the large **H**-matrix, which is needed if diffraction order is 3 or higher, takes a substantial amount of time. The computation time for the source term, $\mathbf{q}_0$, and for the propagation to the sound pressure at the receiver, are practically negligible when there is one source and one receiver. These example results, together with results for different values of `ngauss` can be described by this relation,

$$t_{HOD,cube,ng=16} \approx 0.28 + 0.29(n_{\text{difforder}} - 2) \text{ s}$$

Again, it should be stressed that these numerical values are just sample values, without any repetitions, and for one particular case.

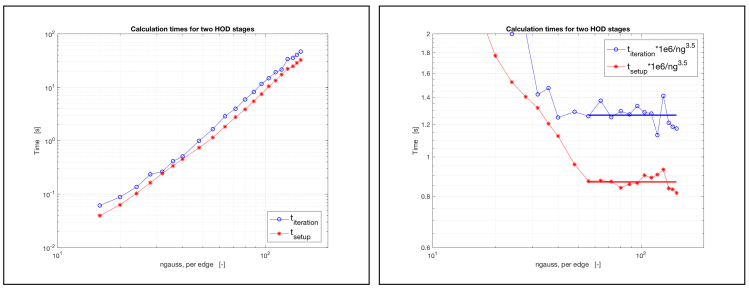
A separate study is made of the influence of number of sources and receivers. Since the **H**-matrix is independent of these two factors, the script `EDexample_cube_HODcost_nSR` has the diffraction order set to 2, and the numbers of sources and receivers are varied. The results for the latter are shown in Fig. 11 (b). It should be noted that the parameter `.doaddsources` was set to 1, which means that all sources' contributions are added up to form a single source term $\mathbf{q}_0$. By repeating for different values of `ngauss`, one finds these approximate relationships,

$$t_{HOD,cube,q_0} \approx 0.26 n_S n_{\text{gauss}} \text{ ms}$$




(a) (b)

Figure 11: Calculation time for higher-order diffraction, at a single frequency, as function of (a) diffraction order, (b) frequency, for 100 source and receiver combinations




(a) (b)

Figure 12: Two components of the calculation time for higher-order diffraction, at a single frequency, as function of $n_{\text{gauss}}$, per edge. (a) Calculation times (b) Calculation times divided by $n_{gauss}^{3.5}$

The number of edge points, or gauss quadrature points, will have a strong influence on the computational cost. The script `EDexample_Cube_HODcost_difforder_ng_batch.m` runs the cube case for a single frequency, one source and one receiver, and varies the number of gauss points per edge from 16 to 148. For each value of gauss points, computations were done for three different diffraction orders, so that the linear slope in Fig. 11 (a) could be estimated. Also, the calculation time for

the setting up of the matrix was averaged for the three diffraction-order calculations. The slope of the two curves in Fig. 12 (a) follows roughly $n_{gauss}^{3.5}$ for the larger values of $n_{gauss}$, as illustrated in Fig. 12 (b). Thus, the set up of the **H**-matrix, and one iteration step, takes roughly similar amounts of time, and in our non-repeated calculation time sample,

$$t_{\text{set}-\text{up } \mathbf{H}} \approx 0.9 n_{\text{gauss}}^{3.5} \mu\text{s}$$

$$t_{\text{one iteration step}} \approx 1.3 n_{\text{gauss}}^{3.5} \mu\text{s}$$

These dependencies can then be gathered in one formula, while adding the fact that all these calculations are done one frequency at a time,

$$\begin{aligned} t_{\text{HOD}} &= n_{\text{freq.}} \left[ t_{\text{set}-\text{up } \mathbf{H}} + n_{\text{iterations}} t_{\text{one iteration step}} + t_{\mathbf{q}_0} + t_{\text{propagate to p}} \right] \\ &= n_{\text{freq.}} \left[ A_1 n_{\text{gauss}}^{3.5} + A_2 (n_{\text{iterations}} - 2) n_{\text{gauss}}^{3.5} + A_3 n_{\text{sources}} n_{\text{gauss}} + A_4 n_{\text{receivers}} n_{\text{gauss}} \right] \\ &= n_{\text{freq.}} n_{\text{gauss}} \left[ \left( A_1 + A_2 (n_{\text{iterations}} - 2) \right) n_{\text{gauss}}^{2.5} + A_3 n_{\text{sources}} + A_4 n_{\text{receivers}} \right] \end{aligned}$$

The numerical values found above are

$$A_1 = 0.9 \cdot 10^{-6}, \quad A_2 = 1.3 \cdot 10^{-6}.$$

$$A_3 = 0.3 \cdot 10^{-3}, \quad A_4 = 0.3 \cdot 10^{-3},$$

Apparantly, the $A_2$-term will dominate unless the number of sources, or receivers, is very large. This speaks for finding efficient iteration stopping criteria, rather than the predetermined value that is implemented today.

# 7 Program structure for EDmain_convexESIE

The main functions `EDmain_xxx` run through a sequence of processing blocks that are partly identical for the different main functions. Tables 11-12 shows this sequence for three of those functions, and they are described briefly below.

Table 11: The processing block sequence of three `EDmain_xxx` functions

| convexESIE | EDmain_ convexESIE_time | convex_time |
|---|---|---|
| EDcheckinputstructs | ← | ← |
| EDreadcad/EDreadgeomatrices | ← | ← |
| EDedgeo | ← | ← |
| EDSorRgeo(.Sdata) | ← | ← |
| EDSorRgeo(.Rdata) | ← | ← |
| EDfindconvexGApaths | ← | ← |
| EDmakefirstordertfs | EDmakefirstorderirs | ← |
| EDed2geo | ← | ← |
| EDinteg_submatrixstructure | ← | — |
| EDintegralequation_convex_tf | EDintegralequation_convex_ir | — |
| — | — | EDfindHODpaths |
| — | — | EDmakeHODirs |

Table 12: The processing block sequence of the `EDmain_convexESIEBEM` function

| EDmain_convexESIEBEM | |
|---|---|
| EDcheckinputstructs | |
| EDreadcad/EDreadgeomatrices | |
| EDgensurfreceivers | |
| EDmain_convexESIE with intermediate surface receivers | |
| Propagate intermediate surface pressure to orignal receivers | |

## 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 3. The data is stored in a struct called `planedata`, see section 10.1 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 section 10.2 for details.

`Input: planedata,geoinputdata.firstcornertoskip`
`Output: planedata, edgedata`

## 7.4 EDSorRgeo

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

`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 section **??**.

`Input: planedata,edgedata, Sdata, Sinputdata.doallSRcombinations,`
`Rdata,controlparameters.difforder,`
`controlparameters.directsound`
`Output: firstorderpathdata`

## 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
```

## 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
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
```

## 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 Ref. [4]. 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
```

## 7.10 EDfindHODpaths

This function is run only if difforder $> 1$. 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 $> 1$. 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
```

# 8 Some auxiliary functions

## 8.1 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 of 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.

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.
```

## 8.2 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.

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

## 8.3 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

coordinates = EDcirclepoints(radius,numberofcircles)
```

## 8.4 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);
```

# 9 Some more example scripts for `EDmain_convexESIE`

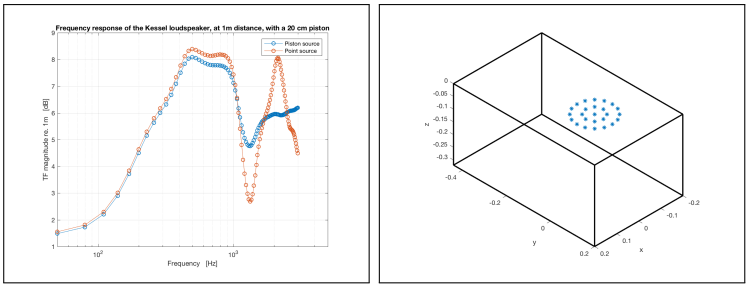
In the folder `EDexamples`, a number of examples of different types are given. These scripts are hopefully easy to modify to individual needs. In Section 2.4, a loudspeaker simulation was demonstrated, with a single point source representing the loudspeaker element.

## 9.1 `EDexample_LspKessel_piston.m`

This example demonstrates how to simulate a circular piston, with the script `EDexample_LspKessel_piston.m`. Below, the parts of this script that differ from `EDexample_LspKessel_minimal.m` are given, and Fig. 13 shows the results, and the model (without the receiver position), respectively. In Fig. 13, also the frequency response from Fig. 4 is shown for comparison.

The auxilliary function `EDcirclepoints` distributes points equally in $n$ concentric circles, and positions them around the origin in the $z = 0$ plane.

```
controlparameters = struct('frequencies',linspace(50,3000,100));
nfreq = length(controlparameters.frequencies);
sources = EDcirclepoints(0.1,2);
sources(:,3) = 0.00001;
nsources = size(sources,1);
Sinputdata = struct('coordinates',sources);
Sinputdata.doaddsources = 1;
Sinputdata.sourceamplitudes = ones(nsources,nfreq)*1/nsources;
```



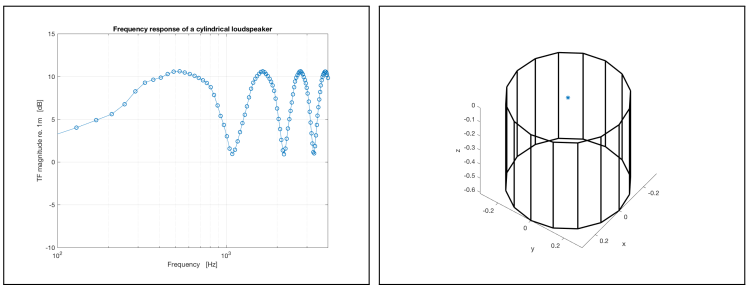

(a) (b)

Figure 13: The "Kessel_piston" loudspeaker example with a 20 cm piston source. (a) The frequency response at 1m distance, for the piston and point sources. (b) The loudspeaker model, with the piston represented by 25 point sources.

## 9.2 EDexample_LspCylinder16.m

The shape of a loudspeaker enclosure has much influence on the frequency response, as shown in [11]. 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. 14 tries to mimic Fig. xx in [11]. 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 [11]. Parts of the script contents are given below. The auxiliary function `EDmakegeo_cylinder` is used to generate the polygonal cylinder shape.

```
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);

Sinputdata = struct('coordinates',[0 0 0.00001]);
Rinputdata = struct('coordinates',[0 0 6.6]);
controlparameters = struct('frequencies',linspace(50,4000,100));
controlparameters.ngauss = 24;
controlparameters.difforder = 30;
```




(a) (b)

Figure 14: 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.

## 9.3 `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. 15 show the impulse responses of the individual diffraction orders, and the corresponding frequency responses (transfer functions), including different numbers of diffraction orders. In Fig. 15 (d), a low-pass filtering effect can be observed for the direct sound. 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 [13], which used an edge diffraction model with a substantial low-frequency error, but with identical results to the method here for higher frequencies.

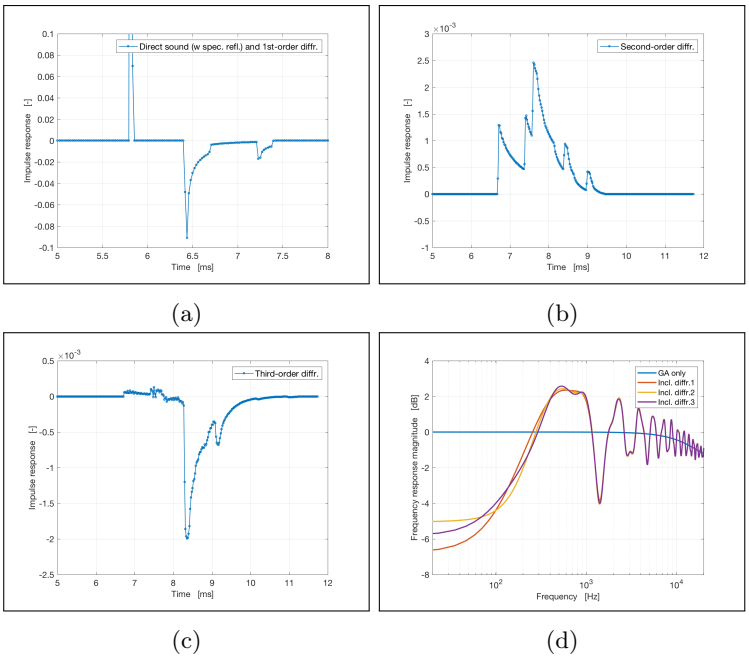
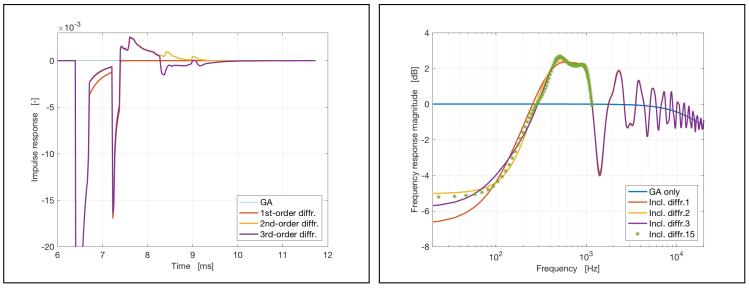



Figure 15: Diffraction impulse responses and frequency responses for the Kessel loudspeaker case, computed with `EDmain_convex_time`. (a) Direct sound and first-order diffraction components. The direct sound, around 5.8 ms, has an amplitude which is not shown, in order to see the weaker first-order diffraction components. (b) Second-order diffraction components. (c) Third-order diffraction components. (d) The corresponding frequency responses.

## 9.4 `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 low range. The script `EDexample_LspKessel_ir_and_tf.m` runs both `EDmain_convex_time` and `EDmain_convexESIE`. The result diagrams in Fig. 16 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.




(a) (b)

Figure 16: Diffraction impulse responses and frequency responses for the Kessel loudspeaker case. (a) Direct sound and first-order diffraction components, computed with `EDmain_convex_time`. The direct sound, around 5.8 ms, is not shown. (b) The corresponding frequency responses, and the frequency response computed with `EDmain_convexESIE`.

# References


[1] , *CATT Acoustic* Gothenburg, Sweden.

[2] U. P. Svensson, R. I. Fred, and J. Vanderkooy, "An analytical edge source interpretation of edge diffraction". J. Acoust. Soc. Am. **133**, 3681–3691 (2013).

[3] R. T. Torres, U. P. Svensson and M. Kleiner, "Computation of edge diffraction for more accurate room acoustics auralization," J. Acoust. Soc. Am. **109**, 600–610 (2001).

[4] A. Asheim and U. P. Svensson, "An integral equation formulation for the diffraction from convex plates and polyhedra," J. Acoust. Soc. Am. **133**, 3681–3691 (2013).

[5] U. P. Svensson, P. T. Calamia and S. Nakanishi, "Frequency-domain edge diffraction for finite and infinite edges," Acta Acustica/Acustica **95**, 568–572 (2009).

[6] U. P. Svensson and A. Asheim, "Time-domain formulation of an edge source integral equation," Proceedings of the International Conference in Acoustics **95**, 568–572 (2009).

[7] U. P. Svensson and P. T. Calamia , "Edge-diffraction impulse responses near specular-zone and shadow-zone boundaries," Acta Acustica/Acustica **92**, 501–512 (2006).

[8] U. P. Svensson, H. Brick and J. Forssén, "Benchmark cases in 3D diffraction with different methods," Proceedings of Forum Acusticum, Krakow, Poland, 7-12 Sep. (2014).

[9] S. R. Martin and U. P. Svensson, "Double diffraction models: a study for the case of non-convex bodies," Proceedings of Forum Acusticum, Krakow, Poland, 7-12 Sep. (2014).

[10] S. R. Martin, U. P. Svensson, J. Slechta, J. O. Smith, "A hybrid method combining the edge source integral equation and the boundary element method for scattering problems," in Proceedings of Meetings of Acoustics **26**, 015001 (2016).

[11] H. F. Olson, "Direct radiator loudspeaker enclosures," J. Audio Eng. Soc. **17**, 22–29 (1969).

[12] L. M. Delves and J. L. Mohamed, *Computational methods for Integral equations* (Cambridge University Press, Cambridge, UK, 1985).

[13] J. Vanderkooy, "A simple theory of cabinet edge diffraction," J. Audio Eng. Soc. **39**, 923–933 (1991).

# 10 Appendix - Internal variables

## 10.1 The planedata struct

Based on the definition of corners and planes as described above, internally in the main program EDmain_convexESIE, the function EDreadcad or EDreadgeomatrices generates the planedata struct, which is passed on internally inside this main function. If the function EDreadcad was used to specify the geometry, then it is possible to set filehandlingparameters .savecadgeofile = 1, and this "cadgeofile" will contain the planedata struct. Table 13 describes the contents of this struct.

Table 13: The planedata struct

| Field name | Size | Values |
|---|---|---|
| .corners | [ncorners,3] | Taken from input data |
| .planecorners | [nplanes,nmax][1] | Taken from input data |
| .planeabstypes | sparse([nplanes,nn]) | [2] |
| .planeeqs | [nplanes,4] | [3] |
| .ncornersperplanevec | [nplanes,1] | No. of corners per plane |
| .minvals | [nplanes,3] | $[\min(x_i), \min(y_i), \min(z_i)]$[4] |
| .maxvals | [nplanes,3] | $[\max(x_i), \max(y_i), \max(z_i)]$[4] |
| .planehasindents | [nplanes,1] | 0 or 1 |
| .indentingcorners | [nplanes,nmax] | 0 or cornernumber [5] |
| .cornerinfrontofplane | [nplanes,ncorners] | -1, 0 or 1 [6] |
| .modeltype | — | 'convex_ext' or 'convex_int' or 'singleplate' or 'thinplates' or 'other' |

[1] The value nmax is the maximum number of corners per plane.
[2] The values are either taken from the cad-file (EDreadcad) or given the value 'RIGID' for each plane (EDreadgeomatrices). The size nn depends on the maximum length of the absorber names that are used in the cad file.
[3] The plane equations are on the form that row $n$ contains the four coefficients $[A, B, C, D]$ on the plane equation form

$$Ax + By + Cy = D$$

where $[A, B, C]$ are normalized to give the plane normal vector.
[4] These min- and max-values give each plane's "axis-aligned bounding box (AABB)"
[5] For each plane this matrix gives 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.
[6] 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

## 10.2 Edges - the edgedata struct

Using the data in the planedata struct, all edges are identified by the function EDedgeo, and data is stored in the struct edgedata. In addition, some more fields are added to the planedata struct. These two structs can be inspected if filehandlingparameters.saveeddatafile is set to 1 as input. Fig. 17 shows the example that corresponds to Fig. ??, and tables 14 and 15 show the additions to the planedata struct and the contents of the edgedata struct.

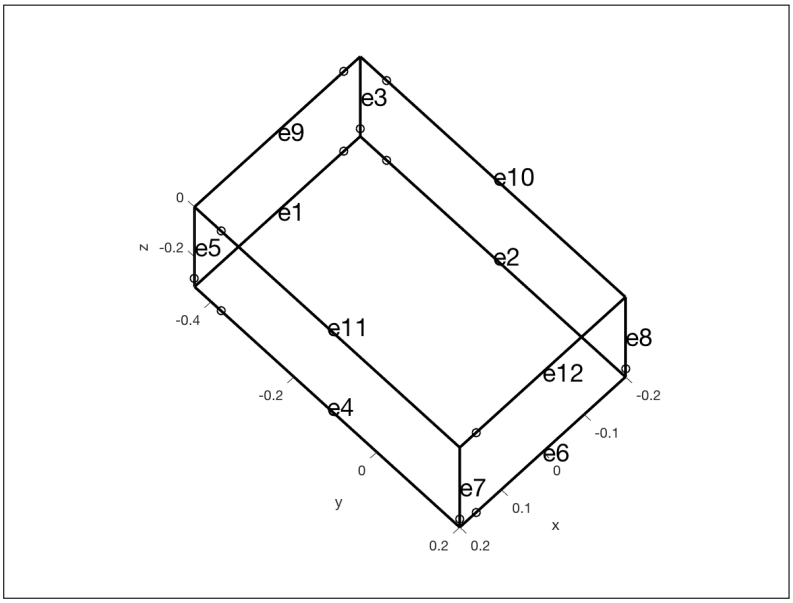



Figure 17: Illustration of a cuboid scattering object, with the derived edge numbers indicated. For each edge, the little circle indicates the starting end.

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

| **Field name** | **Size** | **Values** |
|---|---|---|
| `.planeisthin` | [nplanes,1] | 0 or 1 |
| `.planeseesplane` | [nplanes,nplanes] | -2,-1,0,1 [1] |
| `.rearsideplane` | [nplanes,1] | 0 or planenumber [2] |
| `.planedata.canplaneobstruct` | [nplanes,1] | 0 or 1 [3] |
| `.reflfactors` | [nplanes,1] | -1 (SOFT), 0 (TOTABS), 1 (RIGID) |

[1] These values specify, for each plane-plane pair:
1 means that a plane is in front of the other plane, but obstruction 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
[2] This value is relevant, and non-zero, only for thin planes.
[3] States whether a plane has the potential to obstruct or not (in a plane-to-plane path; which is irrelevant for external convex problems)

Table 15: The `edgedata` struct

| Field name | Size | Values |
|---|---|---|
| `.edgecorners` | [nedges,2] | corner numbers [1] |
| `.closwedangvec` | [nedges,1] | [2] |
| `.edgestartcoords` | [nedges,3] | [3] |
| `.edgestartcoords` | [nedges,3] | [3] |
| `.edgelengthvec` | [nedges,1] | [4] |
| `.offedges` | [noffedges,1] | [5] |
| `.edgenvecs` | [nedges,3] | [6] |
| `.edgenormvecs` | [nedges,3] | [7] |
| `.edgestartcoordsnudge` | [nedges,3] | [8] |
| `.edgeendcoordsnudge` | [nedges,3] | [8] |
| `.edgerelatedcoordsysmatrices` | [nedges,9] | [9] |
| `.indentingedgepairs` | [nn,2] | [10] |
| `.planesatedge` | [nedges,2] | The two connected planes for each edge |
| `.edgesatplane` | [nplanes,nmaxcp][11] | The connected edges for each plane |
| `.edgeseesplane` | [nplanes,nedges] | -2,-1,0 or 1 [12] |

[1] Each edge has a starting corner (column 1) and an ending corner (column 2). This diretion is maintained through all calculations.
[2] For each edge, the "closed wedge angle" is given, in radians. For a 90-degree corner inside a room, the value would be $3\pi/2$. For a 90-degree external corner of a scattering box, the value would be $\pi/2$.
[3] This is redundant; could have been found from `planedata.corners(edgecorners(:,1),:)` (edgestartcoords) or from `planedata.corners(edgecorners(:,2),:)` (edgeendcoords)
[4] This is redundant; could have been computed from the knowledge of the starting and ending coordinates for each edge.
[5] The list gives 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.
[6] Each edge has a reference plane/face, which 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. This matrix gives those normals, for each edge.
[7] `.edgenormvecs` give the normalized vector along the edge
[8] Same as `.edgestartcoords`, but moved a short distance away from the edge endpoint.
[9] For each edge, the 9 values form a matrix which can be used to compute the coordinates of points in the edge-related coordinate system (which is done a real lot). Each row has to be reshaped:
`Bmatrix =reshape(edgerelatedcoordsysmatrices(edgenumber,:),3,3);`
[10] indentingedgepairs
[11] nmaxcp = nmaxcornersperplane = the maximum number of corners for any plane
[12] These values specify, for each plane-edge pair:
1 means that the edge has at least some point in front of the plane, but obstruction has not been checked
0 means that the edge is completely behind the plane
-1 means that the edge is aligned with the plane, but does not belong to it
-2 means that the edge belongs to the plane

## 10.3 The `Sdata` and `Rdata` structs

The two structs have mirrored contents, but with small name changes. These structs primarily contain visibility data, but the fields `Sdata.sources`, and `Rdata.receivers` are copied directly from the input structs `Sinputdata` and `Rinputdata`, respectively. The fields are described in Table 16. For `Rdata`, the contents are corresponding, with obvious name changes.

Table 16: Output data struct `Sdata`

| Field name | Size | Contents |
|---|---|---|
| `.sources` | [nS,3] | S coordinates: [x,y,z][1] |
| `.visplanesfroms` | [nplanes,nS] | 0 - 5 [2] |
| `.vispartedgesfroms` | [nedges,nS] | 0 or $2^n - 1$ |
| `.soutsidemodel` | [1,nsources] | 0 or 1 |
| `.vispartedgesfroms_start` | [nedges,nS] | 0 or 1 |
| `.vispartedgesfroms_end` | [nedges,nS] | 0 or 1 |
| `.reftoshortlistS` | [nedges,nsources] | pointer to the short lists [3] |
| `.rSsho` | [nshortlist,3] | unique values of rS [3] |
| `.thetaSsho` | [nshortlist,3] | unique values of thetaS [3] |
| `.zSsho` | [nshortlist,3] | unique values of zS [3] |

[1] Direct copy from `Sinputdata.coordinates`
[2] Values 0 - 5 imply:
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
[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 | Contents |
|---|---|---|
| `.specreflIScoords` | [nIS,3] | IS coordinates: [x,y,z] |
| `.specrefllist` | [nIS,3] | [S,R,amp] for each row |
| `.diffpaths` | [nreceivers,nsources,nedges] | 0 or 1 |
| `.edgeisactive` | [nedges,1] | 0 or 1 |
| `.directsoundlist` | [ndircombs,3] | [S,R,amp] for each row |
| `.ncomponents` | [1,3] | [ndircombs,nIS,ndiffrcombs] |

Table 18: Output data struct `Hsubmatrixdata`

| Field name | Size | Contents |
|---|---|---|
| `.Hsubmatrix` | [nedgepairs,nedgepairs][1] | |
| `.listofsubmatrices` | [nsub,3] | |
| `.edgepairlist` | [nedgepairs,2] | edgenumbers |
| `.edgetripletlist` | [nsub,3] | edgenumbers |
| `.submatrixcounter` | | nsub |
| `.nuniquesubmatrices` | | nuniquesub |
| `.nedgeelems` | [nedges,1] | |
| `.bigmatrixstartnums` | [nedgepairs,1] | |
| `.bigmatrixendnums` | [nedgepairs,1] | |
| `.reftoshortlist` | [nsub,1] | |
| `.isthinplanetriplet` | [nsub,1] | 0 or 1 |
| `.isthinplaneedgepair` | [nedgepairs,1] | 0 or 1 |
| `.quadraturematrix_pos` | [ngauss,ngauss] | sparse matrix |
| `.quadraturematrix_weights` | [ngauss,ngauss] | sparse matrix |

# 11 Appendix: Notes on the program structure and implementations

Below, some further information is given about the functions described in section 7. The further information is presented in smaller font, while the normal-font text is taken directly from section 7.

## 11.1 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`.

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.

## 11.2 EDmakefirstordertfs

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

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

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

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 Ref. [5]. 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 Ref. [7] is applied for that analytical integration.

## 11.3 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`.

This submatrix structure is further described in the next section, also referring to the struct `Hsubmatrixdata` described in Table 18.

## 11.4 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 Ref. [4]. 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.

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:

$$\mathbf{q} = \mathbf{q_0} + \mathbf{Hq} \tag{2}$$

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. (2) 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.