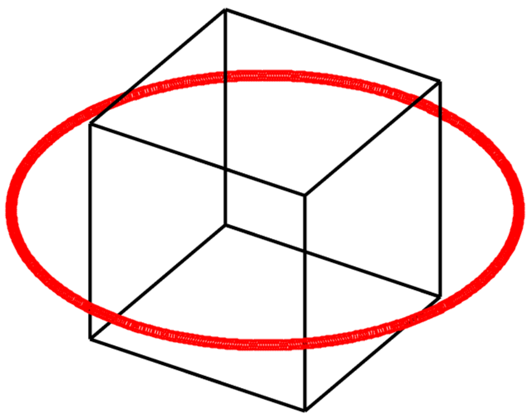
Edge diffraction toolbox
EDtoolbox v 0.1
Manual



Peter Svensson

Acoustics group, Department of Electronic Systems, NTNU

February 8, 2018

1 Introduction

The edge diffraction Matlab toolbox presented here is a somewhat polished version of several previous generations of toolboxes for various edge diffraction computations, as described in the next subsection. In a first version, v 0.1, of this new toolbox called '`EDtoolbox`' a restricted set of problems can be studied: external scattering from convex bodies, in the frequency domain. The edge source integral equation (ESIE) gives remarkably accurate results for convex bodies regardless of frequency, except for certain receiver source/positions where singularities limit the accuracy. One goal for this toolbox is that compact scripts can document precisely how a computation was done, which should help to make computational research 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 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 reflections 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. Thus, it seems 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, the first-order diffraction singularity was finally implemented for the frequency-domain implementation.

1.2 License

TO FIX: SHOULD IT BE GPL OR BSD, AS AT MATYLAB CENTREAL?

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 and Jan Slechta 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

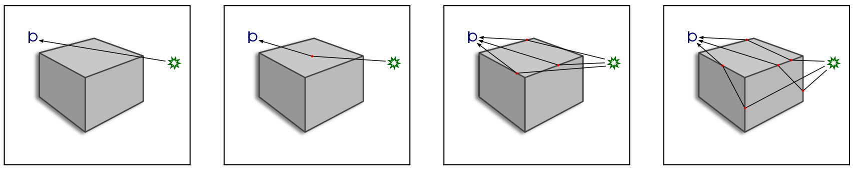
The EDtoolbox computes the scattered, that is, reflected + diffracted, sound pressure for a scattering object which is ensonified by one or more monopole sources. As of version 0.1, frequency-domain formulations are implemented and geometries can be external only. The only supported boundary condition is ideally rigid surfaces (Neumann boundary condition). The sound pressure at a receiver is decomposed according to this model:

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

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[1], 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 four terms in the last formulation in Eq. (1) are computed separately and stored in output files as variables with either the names

`tfdirect`, `tfgeom`, `tfdiff`, `tfinteqdiff`,

for a frequency-domain (FD) calculation, or the names

`irdirect`, `irgeom`, `irdiff`, `irinteqdiff`,

for an upcoming time-domain (TD) implementation. Quite different methods are used for the calculation of these components:

[1]The modification of the direct sound amplitude when it passes exactly through an edge is actually frequency-dependent. As demonstrated in [7], the high-frequency asymptotic value is 0.5, while the low-frequency amplitude depends on the wedge angle. This latter effect results from the contributions from the edge after the instantaneous direct sound component.

- `tfdirect` and `tfgeom` are the geometrical acoustics (GA) terms[2], and they are computed with explicit expressions. The visibility factor, very near the zone boundaries, must be determined precisely.

- `tfdiff` is the first-order diffraction term, which is computed as a sum of explicit integrals, one for each visible edge, which are solved to a high accuracy using Matlab's `QUADGK` function. 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. Furthermore, the integrand becomes singular near those zone boundaries, and here, a frequency-domain version of the approach in [7] is used.

- `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]. 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 $O(n_{\text{gauss}}^2)$, and a minimum of three-four such quadrature points per wavelength[4] is required to keep the 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.

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

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

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 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

[2]Note that in the result file, the variable named `tfgeom` gives only the specular reflection, not the GA solution.

[3]To be precise, the iteration order is the diffraction order - 2.

[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.

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. It is clear that first-order diffraction is done 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, [**?**], 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.

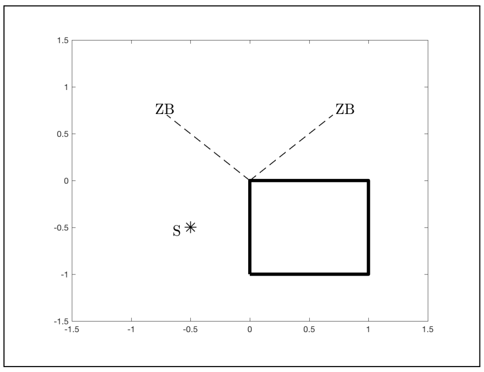


Figure 2: Illustration of the zone boundaries, ZB, for one of the edges of a scattering object, which cause discontinuities for the GA terms, and the first-order diffraction.

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 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

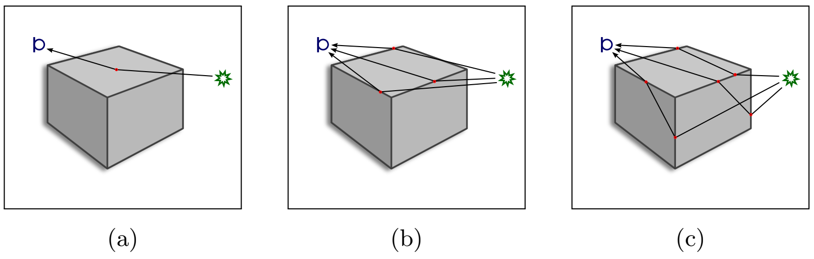


Figure 3: TO DO: DRAW FIGURES
Illustration of the singularities for the higher-order diffraction: (a) Source term, $\mathbf{q_0}$ computation, (b) Iteration for the determining $\mathbf{q}$, (c) Propagation of $\mathbf{q}$ to p at the receiver.

source directivity singularity makes the result converge very slowly, [**?**]. A hybrid method can overcome this, as shown in [**?**]. This hybrid method is not implemented directly in `EDtoolbox`, but can be handled by postprocessing.

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 0.1 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.

2.2 Implemented versions

As of version 0.1 of the toolbox, a restricted set of cases has been implemented, as described in Table 1.

Table 1: `EDtoolbox` - Implemented cases

Geometrical domain	**Frequency-domain**	**Time-domain**
External, convex scattering	`EDmain_convexESIE`	`EDmain_convexESIE_ir` (Coming soon)
Internal problems, or external, non-convex	Not implemented yet	Not implemented yet

The calculation methods implemented are described briefly as follows.

2.3 How to run `EDmain_convexESIE`

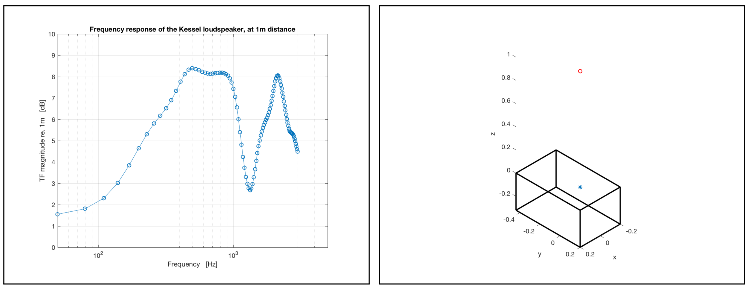
The function is run by assigning values to six input structs, **`geofiledata, Sindata, Rindata, envdata, controlparameters, filehandlingparameters`**, each with a number of fields, that are further described in Section 4.

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. 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 examples 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

geofiledata = struct('corners',corners,'planecorners',planecorners);
Sindata = struct('coordinates',[0 0 0.00001]);
Rindata = 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(geofiledata,Sindata,Rindata,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. (b) The loudspeaker model, with source and receiver positions.

2.4 Known bugs

A few bugs are known, as of v 0.108. 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 x.
- 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 y.

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`.

2.5 Planned developments

- The output data should contain markers for when a singularity was encountered.
- It could be possible to avoid much re-running of calculations by checking all stored files for possible recycling. This can be implemented by storing the relevant setting structs in every result file, final or intermediate, in one directory.
- A time-domain version of the ESIE will be implemented.

- A time-domain version handling of multiple-order diffraction with the "separate order handling" will be implemented. This approach can be very efficient for very low orders of diffraction: up to 2 or 3.
- Some supporting functions might be developed for the hybrid method, for distributing surface receiver points.
- An automatic iteration stop criterion, for the integral equation solving, could quite easily be implemented.
- 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 geofiledata (fields `.corners` and `.planecorners`, as in the example in section 2.3), 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

```
geofiledata.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 corners must be defined in a counter-clockwise order, as seen from the frontal side of the plane. The example in Fig. 5 would have its planes defined as

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

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 `geofiledata.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.

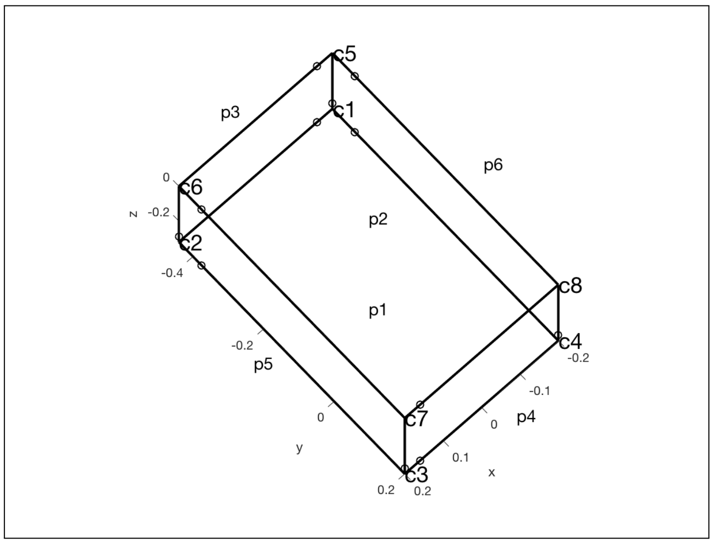


Figure 5: Illustration of a cuboid scattering object. Corner numbers and plane numbers are indicated.

4 Input data

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

```
EDmain_convexESIE(geofiledata,Sindata,Rindata,envdata,...
controlparameters,filehandlingparameters)
```

These six structs are described in Tables 2 - 8.

Table 2: Input data struct `geofiledata`

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 `geofiledata`
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.
[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.

Table 3: Input data struct `Sindata`

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, you 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., if you want to compute backscatter cases, with the receiver in the same location as the source, but do this for a number of incidence angles.

Table 4: Input data struct `Rindata`

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

Table 5: Input data struct `envdata`

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

Table 6: Input data struct `controlparameters`

Field name	**Required?**	**Default value**	**Size, or possible values**
`.docalctf`	—	1	0 or 1 [1]
`.docalcir`	Irrelevant [2]	1	0 or 1
`.frequencies`	Yes [3]	—	[1,nfreq]
`.fs`	Irrelevant [2]	44100	—
`.directsound`	—	1	0 or 1
`.difforder`	—	15	integer >= 0
`.skipfirstorder`	—	0	0 or 1[4]
`.HODcalculation`	Irrelevant [2]	'ESIE'	'ESIE', 'seporders'
`.nedgepoints_visibility`	Irrelevant [5]	2 [6]	—
`.Rstart`	—	0 [7]	—
`.discretizationtype`	—	2	0 or 2 [8]
`.ngauss`	—	16 [9]	even integer >= 2

[1] If the field `.docalctf` 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.
[4] If this parameter is set to 1, the direct sound, specular reflection, and first-order diffraction will not be computed. Please note that a file with the extension `_tf` will still be saved, with empty variables.
[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 default value of 2 implies that the two end points of each edge will be tested for visbility.
[7] The parameter `.Rstart` determines the phase of the final transfer function (or the definition of time zero in upcoming 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.
[8] 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.
[9] The value `.ngauss` specifies the number of quadrature points along the longest edge. It will be scaled down linearly based on the length of each edge, and an even number of quadrature points will always be chosen.

Table 7: Input data struct `filehandlingparameters`

Field name	Required?	Default value	Possible values
`.outputdirectory`	Yes [1]	Same as geoinputfile [2]	—
`.filestem`	Yes[1]	Name of cad-file	—
`.savesetupfile`	—	1	—
`.savecadgeofile`	—	0	—
`.saveSRdatafiles`	—	1	—
`.saveeddatafile`	—	1	—
`.savesubmatrixdata`	—	0	—
`.saveinteqsousigs`	—	0	—
`.loadinteqsousigs`	—	0	—
`.savepathsfile`	—	0	—
`.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 function, `EDmain_convexESIE`, can generate several types of optional output files, as indicated by table 7. There are, however, two files that will always be generated, with the most important output: the TF-terms in Eq. (1). These two files are

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

`<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 2.3:

```
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:

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

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`.

5.2 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 8.

One example of values is given below for the example `EDexample_LspKessel_minimal` in Section 2.3.

```
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 8: 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 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 3071 Hz (2.8 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
```

5.4 Other output quantities

The contents in the various output files are described in table 9, and some of the structs are further detailed in tables 13, 14, and 15 in the Appendix, section 10.

Table 9: Contents of the extra output files

Output file[1]	**Contents**[2]
`_cadgeo`	`planedata`
	`extraCATTdata`
`_Sdata, _Rdata`	`Sdata` or `Rdata`
`_eddata`	`planedata`, `edgedata`
`_paths`	`firstorderpathdata`
`_ISEStree`	For future versions

[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.

6 Results: convergence, accuracy, computational cost, limitations

6.1 Influence of the number of edge points

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 quadrature rule, and then the error follows $O(n^{-2})$, where n is the number of discretization points, [9].

The parameter `controlparameters.ngauss`, with default value 16, determines the 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.

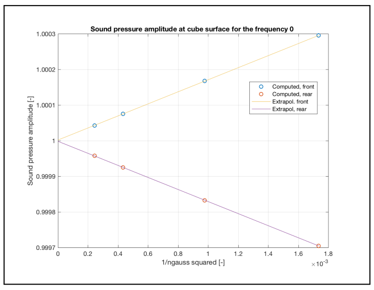
By inspecting the results for the frequency 0, the two receiver points (columns 2 and 3 below), and the four discretizations (see the values in column 1), we see that the results seem to approach 1 as the number of edge points is increased. (`format long` was used to display so many decimals, but we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

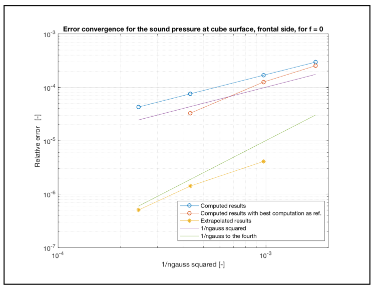
The next step is to apply extrapolation, to illustrate that the error goes as $O(n^{-2})$, where n is the number of discretization points. Thus, the code below generates the result in Fig. 6 (a).

```
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 apparently, 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. 6 (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. Apparently, 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. 6 (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 aparant 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')
```

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. 6 (b). Extrapolated results will be used as reference results, in order to have better estimates of the relative error.

Results were compiled for the script `EDexample_Cube_edgepointsperwavelength.m` as well as for a script `EDexample_Squareplate_edgepointsperwavelength.m` and `EDexample_Octahedrom_edgepointsperwavelength.m`. For each geometry, one receiver was defined without any specular reflections (or direct sound), and one receiver position which had a specular reflection (and direct sound). Furthermore, for each geometry example, 9 discretizations were run: 12, 14, 16, 20, 24, 32, 48, 64, and 72 points per edge. 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. For each of these frequencies, a reference result estimate was first 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. This yielded 540 data points, by combining the three geometries. Fig. 7 shows these data points, together with line fits. Dashed lines indicate the 99% percentiles, and it can be seen that 4 elements per wavelength keeps the error below 1e-2. For the studied cases, apparantly 3 elements per wavelength gives very similar accuracy, but four edge points per wavelength might serves as suitable, conservative rule of thumb.

$$\boxed{\text{A minimum of four edge points per wavelength}}$$

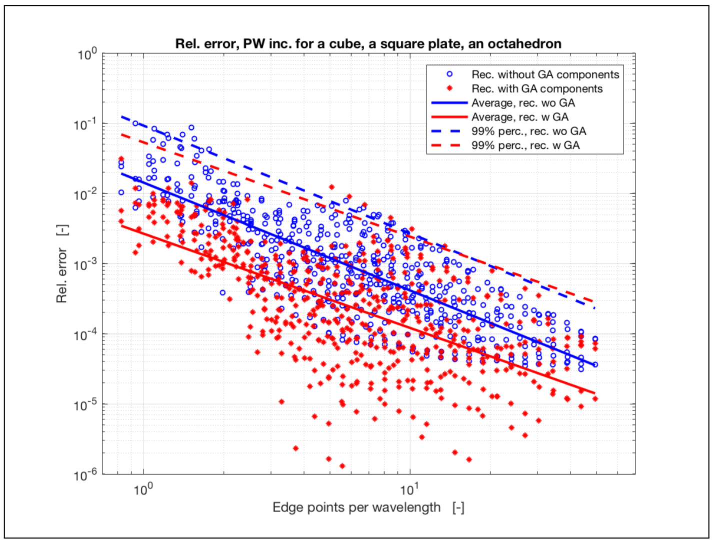


Figure 7: Relative error as function of number of edge points per wavelength. Three geometries were studied: a cube, a square plate and an octahedron. Plane wave incidence was used, and two receivers per geometry case: one with GA components and one without.

6.2 Computational cost

The computational cost can be analyzed using the values in `timingdata`.

6.2.1 First-order diffraction

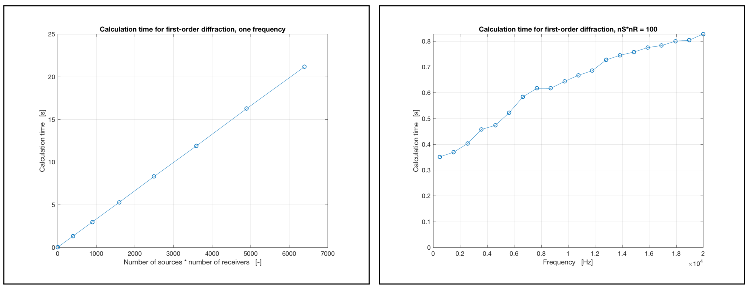
The script `EDexample_Cube_diff1cost.m` runs first-order diffraction calculations for one frequency and a number of sources and receivers. Fig. 8 (a) shows that there is a direct linear dependence on number of sources times number of receivers. The script `EDexample_Cube_diff1costfreq.m` has been prepared to study the influence of the frequency on the first-order diffraction calculation time, and Fig. 8 (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

$$t_{\text{diff.1,cube}} \approx (3,75 + 0,25 f_{\text{kHz}})\, n_S n_R \text{ ms}$$

This expression is interesting mostly for a comparison with the cost for higher-order diffraction.

6.2.2 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. 10 (a). We

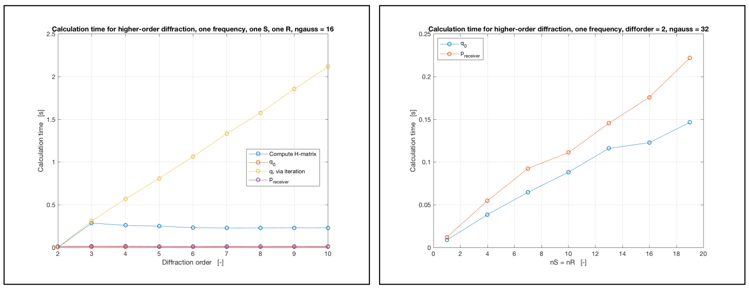


(a) (b)

Figure 8: Calculation time for first-order diffraction as function of (a) number of sources times number of receivers, for one frequency, (b) frequency, for 100 source and receiver combinations

can see that, as expected, the iterative solution part has a constant cost per diffraction order, from order 3. 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 negligible when there is one source and one receiver.

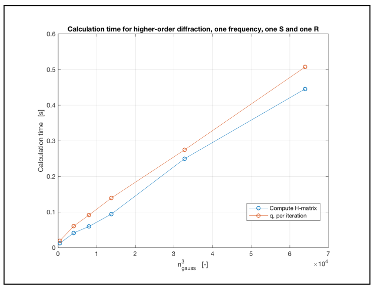
A separate study is made of the influence of number of sources and receivers. Since the **H**-matrix is independent of these two factors, we make a script where the diffraction order is set to 2, and the numbers of sources and receivers are varied, see `EDexample_cube_HODcost_nSR`. The results for the latter are shown in Fig. 10 (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 $\mathbf{q}_0$.



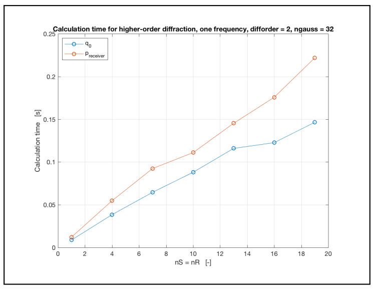
(a) (b)

Figure 9: Calculation time for higher-order diffraction as function of (a) diffraction order, for one frequency, (b) frequency, for 100 source and receiver combinations

The number of edge points, or gauss quadrature points, will have a strong influence on the computational cost. The script `EDexample_cube_HODcost_ng.m` runs the cube case for a single frequency, one source and one receiver, and varies the number of gauss points per edge from 8 to 40. The set up of the **H**-matrix, and one iteration step takes roughly similar amounts of time, and the time for



(a)



(b)

Figure 10: Calculation time for higher-order diffraction as function of (a) n^3_{gauss}, (b) frequency, for 100 source and receiver combinations

this is

$$t_{\text{set}-\text{up } \mathbf{H}} \approx t_{\text{iterate } \mathbf{q} \text{ once}} \sim n^3_{\text{gauss}}$$

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

$$\begin{aligned} t_{\text{HOD}} &= n_{\text{freq.}} \left[t_{\text{set}-\text{up } \mathbf{H}} + n_{\text{iterations}} t_{\text{iterate } \mathbf{q} \text{ once}} + t_{\mathbf{q}_0} + t_{\text{propagate to p}}\right] \\ &= n_{\text{freq.}} \left[A_1 n^3_{\text{gauss}} + n_{\text{iterations}} A_2 n^3_{\text{gauss}} + A_3 n_{\text{sources}} n^2_{\text{gauss}} + A_4 n_{\text{receivers}} n^2_{\text{gauss}}\right] \\ &= n_{\text{freq.}} n^2_{\text{gauss}} \left[\left(A_1 + n_{\text{iterations}} A_2\right) n_{\text{gauss}} + A_3 n_{\text{sources}} + A_4 n_{\text{receivers}}\right] \end{aligned}$$

7 Program structure for EDmain_convexESIE

The main function EDmain_convexESIE runs through the following blocks, in this given order. However, the sequence 7.5 - 7.6 could arbitrarily have been run after the sequence 7.7 - 7.9.

7.1 EDcheckinputstructs

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

```
Input:  geofiledata,Sindata,Rindata,envdata,controlparameters,
        filehandlingparameters
Output:  geofiledata,Sindata,Rindata,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:  geofiledata.geoinputfile
Output:  planedata, extraCATTdata
```

For `EDreadgeomatrices`:
```
Input:  geofiledata.corners,geofiledata.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,geofiledata.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,Sindata.coordinates or
        planedata,edgedata,Rindata.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, Sindata.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,Sindata,
         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`. Clearly, this is needed only if the requested diffraction order > 1.

```
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 sub-matrix 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,Sindata.doaddsources,Sindata.sourceamplitudes,
        Sindata.doallSRcombinations,Rdata,controlparameters,
        filehandlingparameters
Output:  tfinteqdiff,timingdata
```

8 Some auxilliary functions

8.1 `EDplotmodel`

8.2 `EDverify`

8.3 `EDcirclepoints`

8.4 `EDmakegeo_cylinder`

9 Some more example scripts for EDmain_convexESIE

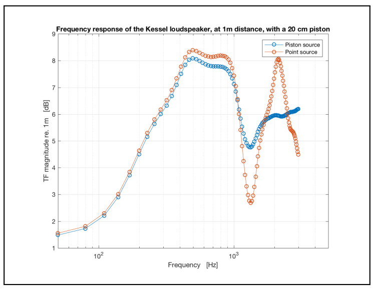
In the folder `EDexamples`, a number of examples of different types are given. These scripts are hopefully easy to modify to individual needs. In Section 2.3, a loudspeaker simulation was demonstrated, with a single point source representing the loudspeaker element.

9.1 `EDexample_LspKessel_piston.m`

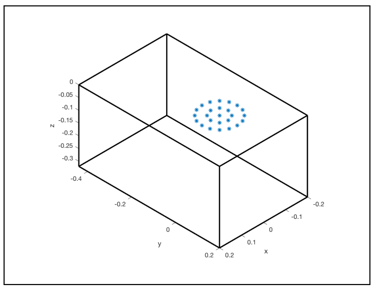
This example demonstrates how to simulate a circular piston, with the script `EDexample_LspKessel_piston.m`. Below, the parts of this script that differ from `EDexample_LspKessel_minimal.m` are given, and Fig. 11 shows the results, and the model (without the receiver position), respectively. In Fig. 11, 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);
Sindata = struct('coordinates',sources);
Sindata.doaddsources = 1;
Sindata.sourceamplitudes = ones(nsources,nfreq)*1/nsources;
```



(a)



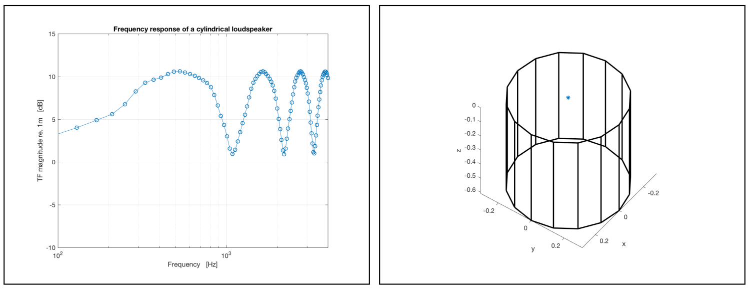
(b)

Figure 11: 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 [8]. 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. 12 tries to mimic Fig. xx in [?]. 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 [?]. 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);
geofiledata = struct('corners',corners,'planecorners',planecorners);

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



(a) (b)

Figure 12: 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 Estimate the error

By running the higher-order diffraction calculations for two or more different discretizations, one can try to apply extrapolation to get an estimate of the final solution. That estimate also leads to an estimate of the error of the previous solution.

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

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

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 10 describes the contents of this struct.

Table 10: 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. 13 shows the example that corresponds to Fig. **??**, and tables 11 and 12 show the additions to the `planedata` struct and the contents of the `edgedata` struct.

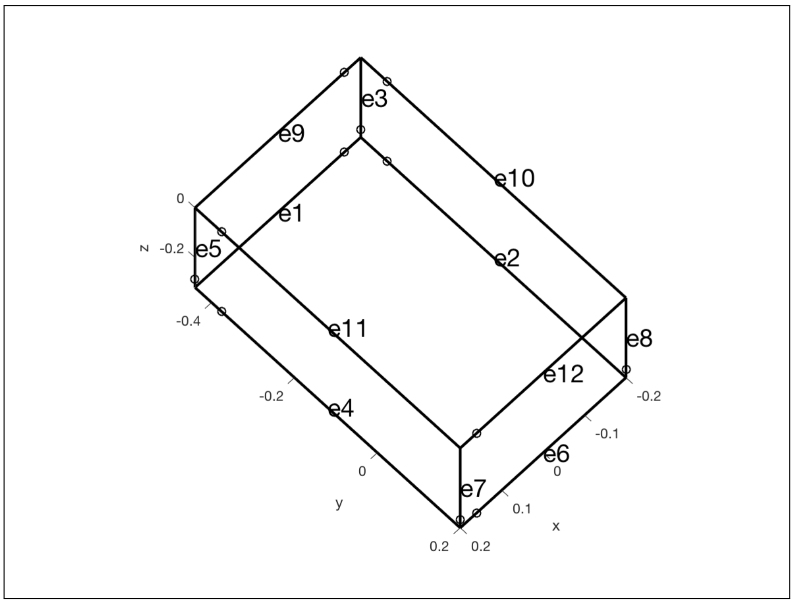


Figure 13: Illustration of a cuboid scattering object, with the derived edge numbers indicated. For each edge, the little circle indicates the starting end.

Table 11: 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 12: 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

Table 13: 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 `Sindata.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 14: 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 15: 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

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. [?]. 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 15.

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 15. 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.