EDtoolbox manual, v 0.1

Peter Svensson, NTNU

January 31, 2018

1 Introduction

The EDtoolbox computes the scattered, that is, reflected + diffracted, sound pressure for a scattering object which is ensonified by one or more monopole source(s). Both time-domain and frequency-domain solutions are possible, and geometries can be external or internal. The only supported boundary condition is ideally rigid surfaces (Neumann boundary condition). Generally, the sound pressure at a receiver is decomposed according to this model:

$$\begin{aligned} p_{\text{total}} &= p_{\text{direct}} + p_{\text{GA}} + p_{\text{diff 1}} + p_{\text{diff 2}} + p_{\text{diff 3}} + \ldots \\ &= p_{\text{direct}} + p_{\text{GA}} + p_{\text{diff 1}} + p_{\text{HOD}} \end{aligned} \tag{1}$$

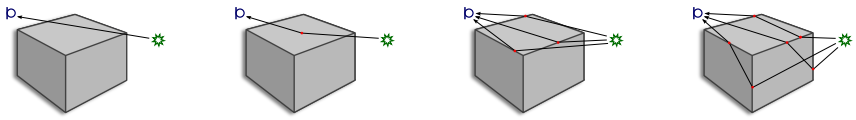
where HOD stands for higher-order diffraction. See Figur 1 for an illustration of some of these terms. It should be noticed that the direct sound component is subject to a visibility test, which means that it is either exactly the same as the free-field sound pressure, or zero,

$$p_{\text{direct}} = p_{\text{free-field}} \cdot V_{\text{S,R}}$$

where

$$V_{\text{S,R}} = \begin{cases} 1, & \text{if the line from S to R is unobstructed} \\ 0, & \text{if the line from S to R is obstructed} \end{cases}$$

Also the other components in Eq. (1) are subject to similar visibility tests.



(a) The direct sound component (b) Specular (geometric) reflection (c) First-order diffraction (d) Second-order diffraction

Figure 1: Illustration of the decomposition of the sound field into components.

The four terms in the last formulation in Eq. (1) are 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 a time-domain (TD) calculation. It should be pointed out that EDtoolbox does not give the value of the sound pressure at a receiver; instead it gives the transfer function, in order to be independent of source amplitude/signal. The transfer functions 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. Equivalently, it can be interpreted that the toolbox 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. So, if one prefers, it is 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.

1.1 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

1.2 How to run `EDmain_convexESIE`

The program is run by assigning values to six input structs, `geofiledata`, `Sindata`, `Rindata`, `envdata`, `controlparameters`, `filehandlingparameters`, further described in Section 3.

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 10 m away. After the calculations have been run, the result files are loaded and a

frequency response is plotted. More details are given in the following sections. 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. Figure **??** shows the resulting output.

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

%-----------------------------------------------------------------------
% Define the scattering object = the 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 10]);
controlparameters = struct('frequencies',logspace(log10(50),log10(3000),100));
filehandlingparameters = struct('outputdirectory',infilepath);
filehandlingparameters.filestem = filestem;

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

EDmain_convexESIE(geofiledata,Sindata,Rindata,struct,controlparameters,...
filehandlingparameters);

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

eval(['load ',infilepath,filesep,'results',filesep,...
filehandlingparameters.filestem,'_tfinteq.mat'])
eval(['load ',infilepath,filesep,'results',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')
grid
```

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

2 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 the previous section), or in a separate file of the .cad-format, which is a format exported by the CATT-Acoustic software. Figure 2 shows a simple example: a cuboid box.

2.1 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. 2, 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; ...
```

2.2 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. 2 would have its planes defined as

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

2.3 The cadfile format

The cadfile format is a very simple format defined by the CATT-Acoustic software. It is a textfile with four sections, marked with textlines `%CORNERS`, `%PLANES`, `%SOURCES`, `%RECEIVERS`. For the use in the EDtoolbox, only the first two are used. Thus, the two sections should have the format given below, exemplifying for the same box as in Fig. 2 (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
```

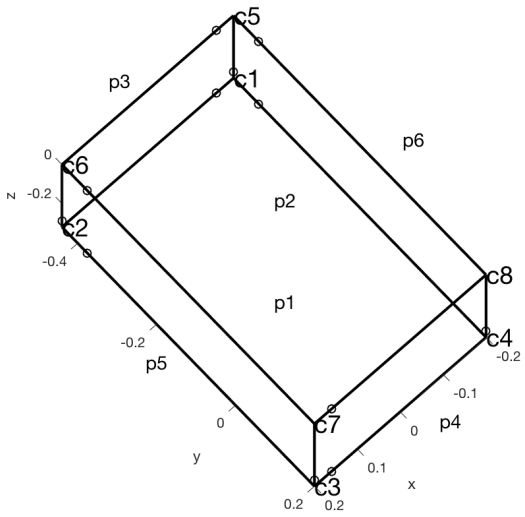


Figure 2: Illustration of a cuboid scattering object. Corner numbers and plane numbers are indicated.

2.4 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 2 describes the contents of this struct.

Table 2: 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

2.5 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. 3 shows the example that corresponds to Fig. 2, and tables 3 and 4 show the additions to the `planedata` struct and the contents of the `edgedata` struct.

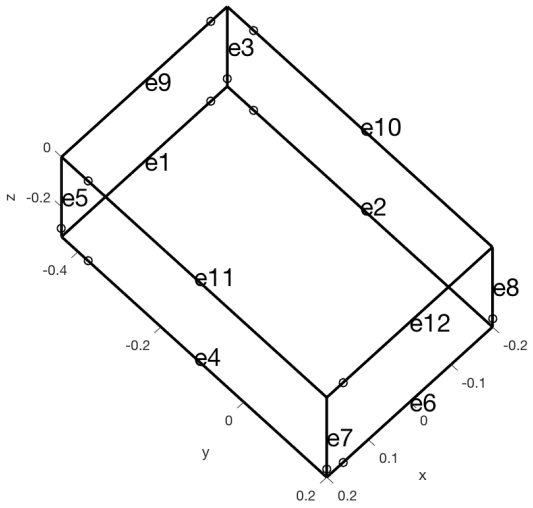


Figure 3: Illustration of a cuboid scattering object, with the derived edge numbers indicated. For each edge, the little circle indicates the starting end.

Table 3: 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 4: 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

3 Input data

The main program, 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 5 - 10.

Table 5: 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 2 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 6: 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 4 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 7: Input data struct `Rindata`

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

Table 8: Input data struct `envdata`

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

Table 9: 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
`.HODcalculation`	Irrelevant [2]	'ESIE'	'ESIE', 'seporders'
`.nedgepoints_visibility`	Irrelevant [4]	2 [5]	—
`.Rstart`	—	0 [6]	—
`.discretizationtype`	—	2	0 or 2 [7]
`.ngauss`	—	16 [8]	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] 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.
[5] The default value of 2 implies that the two end points of each edge will be tested for visbility.
[6] 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.
[7] 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.
[8] 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 10: 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.

4 Output data

4.1 The resulting sound pressure/transfer functions

The main program, EDmain_convexESIE, can generate a lot of output data, as indicated by table 10. 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 1.2:

```
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.

4.2 Other output quantities

The contents in the various output files are described in table **??**, and some of the structs are further detailed in tables 12, 13, and 14.

Table 11: 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.

Table 12: 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 13: 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 14: 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

5 Program structure for EDmain_convexESIE

The main function EDmain_convexESIE runs through the following blocks, in this given order. However, the sequence 5.5 - 5.6 could arbitrarily have been run after the sequence 5.7 - 5.9.

5.1 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
```

5.2 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 2. They are stored in a struct called `planedata`.

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

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

5.3 EDedgeo

This function identifies all the edges of the polyhedron, and stores data about them in a separate struct called `edgedata`.

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

5.4 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.

```
Input:  planedata,edgedata,Sindata.coordinates or
        planedata,edgedata,Rindata.coordinates
Output:  Sdata or
         Rdata
```

5.5 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`.

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

5.6 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
```

5.7 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
```

5.8 EDinteg_submatrix

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

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

5.9 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`.

```
Input:  envdata,planedata,edgedata,edgetoedgedata,Hsubmatrixdata
        Sdata,Sindata.doaddsources,Sindata.sourceamplitudes,
        Sindata.doallSRcombinations,Rdata,controlparameters,
```

```
        filehandlingparameters
Output:  tfinteqdiff,timingdata
```

6 Some examples for EDmain_convexESIE

In the folder `EDexamples`, you will find some examples of different types. These scripts are hopefully easy to modify to your needs.