Acoustic Simulation TOol

User's Manual

v2.0

Umberto Iemma & Vincenzo Marchese
http://acousto.sourceforge.net

Contents

CHAPTER 1

Getting started with AcouSTO

This is where you learn about system setup and AcouSTO *installation*

This chapter deals with the procedure to get AcouSTO working on your machine. The code has been developed and heavily tested on Linux and OSX machines using standard development tools and configuration structure. In principle, AcouSTO should run without troubles on any machine equipped with a UNIX–like OS and meeting the prerequisites (see Section 1.1.1).

A native version for Microsoft Windows is not available. A Cygwin environment should be enough to compile and run the code (but we have never tested this possibility). Windows users can modify the Makefile themselves to make AcouSTO compile and run natively in a Windows environment. In this case, they must keep in mind that they are on their own.

Starting from AcouSTO v2.0, Windows users can easily run AcouSTO within a dedicated Docker container, as described in Section 1.2.

1.1 Native installation

This section deals with the native installation of AcouSTO on UNIX–like OSs. The procedure is extremely simple, since our code has been developed by making use of standard languages and libraries, freely available for all the platforms and architectures.

1.1.1 Prerequisites

The basic prerequisites needed to compile AcouSTO are:

- a C compiler (gcc suggested);
- the complete GNU configure and build system;
- a working MPI-2 implementation (OpenMPI strongly suggested).
- libconfig downloadable from: http://www.hyperrealm.com/libconfig/
- lapack and blas libraries.
- Scalapack and BLACS libraries.

If you want to enable the MySQL save feature you must have libmysqlclient installed (version >= 5.0).

Some optional tools are required for data visualization:

- a VRML and VTK renderer application such as Mayavi or Paraview;

- a 2D `xy` plotting program (*e.g.*, `gnuplot` or `grace`).

All the software needed to run `AcouSTO` has been included into the official repositories of most of the latest Linux distros, making the installation a straightforward procedure.

1.1.2 Installation procedure

The code is released with a standard GNU–compliant configuration file. It can be installed in a system-wide fashion, to make it available to all users, or by a single user for individual use.

1. download the gzipped tar file from the repository;
2. become superuser;
3. unpack it wherever you like (let's say `/usr/local/tmp`);
4. `cd` to the `AcouSTO` directory (`/usr/local/tmp/acousto` in our case);
5. issue the command `./configure --prefix=exec_dir`, where `exec_dir` is the location where you have choosen to install the executable (the directory `exec_dir/bin` must be included in your `$PATH`). Add to the command all the configuration options that fit to your needs. The complete list of available options is
   - `--with-mpicc=<mpicc compiler>` location of `mpicc` compiler
   - `--with-libconfig=<config lib dir>` location of libconfig lib
   - `--with-libmysqlclient=<mysqlclient lib dir>` location of `MySQL` client lib
   - `--enable-mpinew` to link to openMPI ≥ 1.7
   - `--with-blacs=<dir>` location of BLACS libs
   - `--with-lapack=<dir>` location of Lapack libs
   - `--with-atlas=<dir>` location of Atlas libs
   - `--with-scalapack=<dir>` location of Scalapack libs
   - `--with-blas=<dir>` location of BLAS libs
   - `--enable-scalapack_new` to link to Scalapack version ≥ 2.0
   - `--with-intel-libs=<dir>` to be used together with `--enable-intel`
   - `--with-libs=<dir>` if all your libraries are installed in the same place, you can use only this options.
   - `--enable-intel` builds with intel compiler (default is gcc).
   - `--enable-mysql=<yes|no>` if "yes" enables the `MySQL` save feature (see Appendix D). Default is "no".

6. compile and install `AcouSTO` with `make` followed by `make install`

If, for example, you have the files[1]

- liblapack.a
- libscalapack.a
- libblacsCinit_MPI.a
- libblacs_MPI.a

in the directory `/usr/local/mylibs`, you can start the configuration issuing the command:

```
configure --with-scalapack=/usr/local/mylibs --with-blacs=/usr/local/mylibs \
          --with-lapack=/usr/local/mylibs
```

or, equivalently,

```
configure --with-libs=/usr/local/mylibs
```

Users of libconfig version earlier than 1.3.2 must define the flag `HAVE_OLD_LIBCONFIG` issuing the command

- **`export CFLAGS=-DHAVE_OLD_LIBCONFIG` in bash**
- **`setenv CFLAGS -DHAVE_OLD_LIBCONFIG` in csh**

before starting the configuration of the code.

[1] Note that the `AcouSTO` configuration script is able to recognise most of the possible names given to the BLACS libraries in the different environments. Nevertheless, very special installations might need some adaptation. In most of the case, a typical workaround is to establish symbolic links to the libraries using standard names, such the ones we are using here.

In version 1.6, two new important configuration options have been introduced.

1. If the installed version of OpenMPI is $\geq$ 1.7 the name of the FORTRAN interface library has changed. To compile AcouSTO against the newer versions of OpenMPI use the configuration options
   `--enable-mpinew`
2. If Scalapack version is $\geq$ 2.0 the BLACS libraries are embedded into the scalapack file. Now AcouSTO can be linked directly to the new Scalapack using the configuration option
   `--enable-scalapack_new`

1.2 The AcouSTO Docker container

In AcouSTO version 2.0 has been introduced the possibility to build a Docker container from a Dockerfile to run a virtual machine ready for standard AcouSTO use on all the architectures supported. The building of the Docker image is extremely simple and the resulting container runs flawlessly under all host OSs. The advantage of such an approach

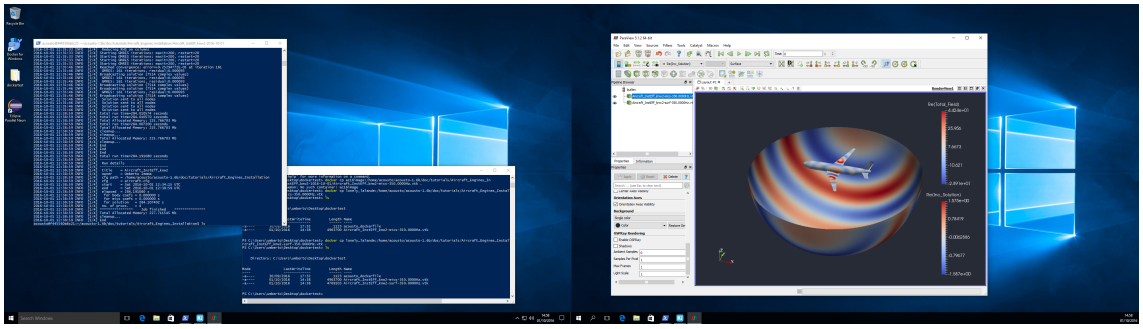


Figure 1.1: AcouSTO Docker container running under Windows 10, with data visualization using Windows–native version of Paraview.

with respect to the standard virtualization based on the full installation of a guest OS onto a virtualization environment is in the extreme lightweight of the underlining software layers. Docker containers use less RAM, start almost instantly and have a much more efficient use of the resources (http://www.docker.com for details). Using the Docker container, AcouSTO can be run on all OSs with the same straightforward installation procedure. Figure 1.1 shows AcouSTO running within the Power Shell of Windows 10, with the Windows–native version of Paraview used to visualise the results. Two procedures can be followed to run the AcouSTO container: *i)* build the Docker image on your system using the acousto dockerfile; *ii)* run directly the prebuilt image from the Docker Hub. The two procedures yield exactly the same result, so you shouldn't expect better performances

from the locally–built image. The choice depends on whether you are interested or not in the development of your personal images by modifying the dockerfile, or simply by your personal attitude in doing things by yourself. Whatever procedure you choose, you must first install Docker on your system. Go to `http://www.docker.com` and follow the installation instructions. Very simple and fast. You shouldn't have problems. After this, follow the instructions below.

Build and run the image from the dockerfile

Four steps are needed to build locally an image of the AcouSTO container.

1. Download the file `acousto_dockerfile` from the AcouSTO website and save on your hard disk. The dockerfile location is totally irrelevant to the success of the building process. The dockerfile contains the Docker commands needed to build the AcouSTO image from the Debian official one.
2. Open a terminal (under OSX or Linux), the command prompt under Windows 7, or the Power Shell under Windows 8, 8.1 or 10. You can also use the interfaces installed with Docker (Kinematics, Docker terminal ...).
3. Change current directory to the one where the `acousto_dockerfile` is and give the command

   ```
   docker build -t name-of-acousto-image -f ./acousto_dockerfile .
   ```

   The command builds the docker image with the name `name-of-acousto-image` following the sequence of commands in the dockerfile. It seems that, at least in my systems, image tag can include only numbers and lowercase letters. It takes few minutes to complete. Warnings will appear during the process. Just ignore them.
4. Run the built image with the command

   ```
   docker run -t -i name-of-acousto-image /bin/bash
   ```

   You are now logged into the AcouSTO virtual machine as user `acousto` (password `acousto`), under the path `$HOME/acousto-X.Y` where `X.Y` is the installed version of AcouSTO . User `acousto` has superuser privileges.

Run a pre–built image from Docker Hub

If you prefer to download a pre-built image from the Docker Hub, you have to open a terminal (or command prompt, or power shell, depending on the hosting OS) and give the command

```
docker run -t -i uiemma/acousto:image-tag /bin/bash
```

where `uiemma/acousto` is the name of the `AcouSTO` repository on the Docker Hub, and *image-tag* points to the specific image you are interested in. The format of *image-tag* is `vXYdistro`, where X.Y is the version of `AcouSTO` installed in the image and `distro` is the name of the Linux distribution on which the image is based. So, if you want to run the Debian–based image installing version v1.6 of `AcouSTO` then *image-tag*=`v16debian`. The initial login is the same as in point 4 of the previous section.

Work inside the AcouSTO container

At container startup you are dropped into a bash, under the path `$HOME/acousto-X.Y`, *i.e.*, the directory containing the `AcouSTO` vX.Y distribution. You can now work freely as you do in any Linux box. You can create directories and move around them in the usual way. You can also install new software from the standard repositories of the Linux distro on which the container is based. Just remember that all the changes that you apply to your running image will be lost at the image shutdown. If you want to make permanent changes you have to save your container into your own image using the `docker commit` command. Follow the tutorial at `https://docs.docker.com/engine/tutorials/dockerimages/`.

You can share data between the container and the host OS in various way. The easiest is to simply copy the files you want to make available to the host OS using the `docker cp` command. While the container is running, open a shell in the host and give the command

```
docker cp container-name:/path/to/your/file .
```

This will save a copy of the file in the host OS current directory. You can also share entire filesystems by mounting volumes of the host OS in your container. The procedure depends strongly on your operating system and we address to the Docker website for details.

On a multicore machine you can exploit multithreading within the `AcouSTO` container, provided that Docker has been properly configured. On OSX and Windows 10 this can be easily done using the Docker application installed. On earlier Windows systems, you have to configure the underlining virtualization environment (typically, Oracle Virtual Box). The performance that you can obtain using OSX, Linux or Windows 10 hosts are aligned with that obtainable from native compilation. On earlier Windows versions the performance strongly depends on the efficiency of the virtualization system. We did some test only on Windows 7 running Oracle Virtual Box in its default configuration. Results are presented in Figure 1.2. The test has been run having configured 8 cores for the Docker machine using the Docker app on OSX and Windows 10 and Virtual Box on Windows 7. The results show that the speed-up achieved by `AcouSTO` rurrning in the Debian–based Docker container are aligned with the corresponding performance of the natively compiled

version, with the only exception of Windows 7 host OS. These results shouldn't be used as a basis for the assessment of the quality of one platform w.r.t. another, but only as a comparison of what can be obtained by an average user with the default configurations of the software. Better performances can be likely obtained through a fine tuning of the virtualization environment, not addressed here.

Docker images can be also used on large clusters basd on MPI2. AcouSTO can take advantage from this feature to setup (in user space) a HPC environment with a limited effort. The interested user is addressed to the Docker website to learn how to do. Other useful references can be found, for instance, at

- http://qnib.org/mpi-paper
- http://arxiv.org/pdf/1509.08231
- http://insidehpc.com/2015/08/video-docker-for-hpc-in-a-nutshell/

We do not provide specific instructions about this point, since they strongly depend on the specific system configurations.

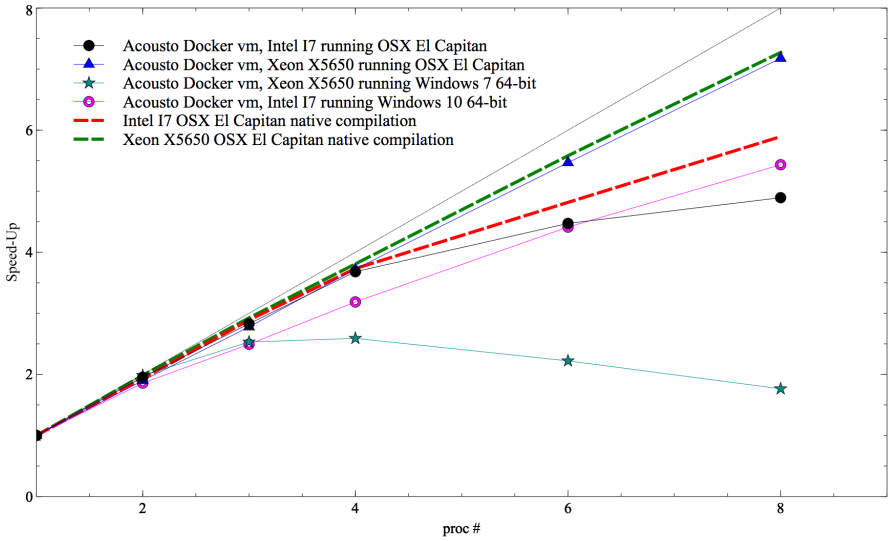


Figure 1.2: AcouSTO multithreading speed-up of the Debian–based AcouSTO Docker VM running under different OS for two CPU architectures. Thick dashed lines refer to standard AcouSTO native compilation under OSX El Capitan for Intel I7 (red) and Xeon X5650 (green).

CHAPTER 2

AcouSTO - A general overview

This is where we tell you about what AcouSTO *is*

In this chapter, a general description of the code is given, to help you understand whether or not AcouSTO could fit to your needs.

2.1 What AcouSTO is

AcouSTO is a boundary element solver solution of the Kirchhoff–Helmholtz Integral Equation (KHIE). The problem is written in the Laplace domain, and reduces to the classical single-frequency formulation along the imaginary axis of the Laplace plane (*i.e.*, for $s = j\,\omega$). The solution is based on a low–order Boundary Element Method (BEM).

2.1.1 The basic equations

The acoustic problem is written in terms of the velocity potential function φ, in the Laplace domain, to yield

$$\nabla^2 \tilde{\varphi}(\mathbf{x}) - \kappa^2 \tilde{\varphi}(\mathbf{x}) = \tilde{q}, \qquad \text{for } \mathbf{x} \in \mathcal{V} \tag{2.1}$$

where the $\tilde{\bullet}$ indicates Laplace transformation, whereas $\tilde{q}$ represents the acoustics sources present in the field, and $\kappa = s/c_0$ is the complex wave number, being $s = \alpha + j\,\omega$ the Laplace variable, and c_0 the speed of sound in the reference conditions. The formulation of the problem in the Laplace domain is equivalent to the time convention $e^{j\,\omega\,t}$. This point must be kept in mind especially when comparing the results with reference analytical solutions. The above problem is completed by suitable boundary conditions for $\mathbf{x} \in \partial\mathcal{V}$ (see Section 4.2). Note that Eq. 2.1 holds for both the velocity potential and the pressure perturbation. The physical meaning of the solution depends on the context, and is left to the user to take care of this, by providing the appropriate boundary conditions. When the problem is addressed in terms of velocity potential, the pressure perturbation $p = P - P_0$ can be obtained from the linearized Bernouilli's theorem, $\tilde{p} = -s\,\varrho\,\tilde{\varphi}$ by simply post-processing the solution. In most of the examples presented in this document, $\tilde{\varphi}$ is assumed to be the velocity potential. Using the standard procedure, and assuming $\tilde{q} = 0$,[1] the boundary integral formulation for the velocity potential has the form

$$E(\mathbf{y})\,\tilde{\varphi}(\mathbf{y}) = \oint_{\mathcal{S}} \left(G \frac{\partial \tilde{\varphi}}{\partial n} - \tilde{\varphi} \frac{\partial G}{\partial n} \right) d\mathcal{S}(\mathbf{x}). \tag{2.2}$$

where the domain function $E(\mathbf{y})$ is

$$E(\mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{y} \in \mathcal{V}, \\ 1/2 & \text{if } \mathbf{y} \in \partial\mathcal{V}, \\ 0 & \text{if } \mathbf{y} \notin \mathcal{V}, \end{cases} \tag{2.3}$$

and $\mathcal{S} = \partial\mathcal{V}$ for internal problems, and $\mathcal{S} = \partial\mathcal{V} \backslash \mathcal{S}_\infty$ for external problems.

[1] The effect of the field sources, and how it is managed in AcouSTO will be discussed in Section 4.2

Recalling the expression of the fundamental solution of Eq.2.1, G,

$$G(\mathbf{x},\mathbf{y},s) = \frac{-e^{-s\,\theta}}{4\pi r} = G_0\, e^{-s\,\theta} \qquad \text{with } r = \|\mathbf{x}-\mathbf{y}\|, \text{ and } \theta = \frac{r}{c_0}, \tag{2.4}$$

Eq. 2.2 becomes the familiar KHIE,

$$E(\mathbf{y})\,\tilde{\varphi}(\mathbf{y}) = \oint_{\mathcal{S}} \left(G_0 \frac{\partial \tilde{\varphi}}{\partial n} - \tilde{\varphi} \frac{\partial G_0}{\partial n} + s\,\tilde{\varphi} G_0 \frac{\partial \theta}{\partial n} \right) e^{-s\,\theta}\, d\mathcal{S}(\mathbf{x}). \tag{2.5}$$

Equation 2.5 is used in AcouSTO to solve the BVP starting from the boundary conditions, and to represent the acoustic field once that the solution on the boundary is known. Specifically, when $\mathbf{y} \in \partial\mathcal{V}$, Eq. 2.5 yields a integro-differential equation for the unknown $\tilde{\varphi}$ on the boundary $\partial\mathcal{V}$, which can be solved from the knowledge of the boundary conditions $\tilde{\chi} = \partial\tilde{\varphi}/\partial n$; once that the solution is known on $\partial\mathcal{V}$, Eq. 2.5 is used as a boundary integral representation for $\tilde{\varphi}$ at arbitrary points in the field (the "microphones") as a function of the known distributions of $\tilde{\varphi}$ and $\tilde{\chi}$ over the boundary.

2.1.2 The numerical solution

The Boundary Integral Equation 2.5 is numerically solved through a Boundary Element Method (BEM). The boundary of the domain is partitioned into N quadrilateral panels, and all the quantities are considered to be constant within each panel (*zeroth–order* approximation). The surface integral in Eq. 2.5 is approximated with a sum of N panel integrals, and the collocation method is used, by locating the collocation points at the centers of the panels. The discrete approximation of Eq. 2.5 is

$$\frac{1}{2}\,\tilde{\varphi}_n = \sum_{m=1}^{N} \left[B_{nm}\,\tilde{\chi}_m + (C_{nm} + s\,D_{nm})\,\tilde{\varphi}_m \right] e^{-s\,\theta_{nm}}, \qquad n = 1,\ldots,N, \tag{2.6}$$

where the subscripts indicate the evaluation at the corresponding collocation point, $\tilde{\chi} = \partial\tilde{\varphi}/\partial n$, and the integral coefficients have the form

$$B_{nm} = \int_{\mathcal{S}_m} G_0\, dS, \quad C_{nm} = \int_{\mathcal{S}_m} \frac{\partial G_0}{\partial n}\, dS, \quad D_{nm} = \int_{\mathcal{S}_m} G_0 \frac{\partial \theta}{\partial n}\, dS. \tag{2.7}$$

In AcouSTO , the coefficients 2.7 are evaluated analytically (details will be provided in the advanced user and programming manual, to be published soon). Collecting the integral coefficients in Eq. 2.7 into the $[N \times N]$ complex matrices B, C, and D, and the value of the velocity potential and its normal derivative respectively into the $[N \times 1]$ column vectors $\underline{\tilde{\varphi}}$ and $\underline{\tilde{\chi}}$, the solution of the system in matrix form is

- For a Neumann BVP ($\underline{\tilde{\chi}}$ known on the boundary)

$$\underline{\tilde{\varphi}} = \mathsf{Y}^{-1}\,\mathsf{B}\,\underline{\tilde{\chi}} \tag{2.8}$$

- For a Dirichlet BVP ($\underline{\tilde{\varphi}}$ known on the boundary)

$$\underline{\tilde{\chi}} = \mathsf{B}^{-1}\,\mathsf{Y}\,\underline{\tilde{\varphi}} \tag{2.9}$$

where, in both cases,

$$\mathsf{Y} = \left(\frac{1}{2}\,\mathsf{I} - \mathsf{C} - s\,\mathsf{D}\right) \tag{2.10}$$

The form of the solving system actually implemented in the code is reported in Section 4.2, where the treatment of the boundary conditions is addressed in detail.

Once the solution of the problem is known on the boundary from Eq. 2.8 or 2.9 , the same integral formulation is used to evaluate the pressure complex amplitude at the M microphones. Collecting the value of $\tilde{\varphi}$ at the microphones into the $[M \times 1]$ column vector $\underline{\tilde{\varphi}}^{\mathsf{M}}$, yields

$$\underline{\tilde{\varphi}}^{\mathsf{M}} = \mathsf{B}^{\mathsf{M}}\,\underline{\tilde{\chi}} + \left(\mathsf{C}^{\mathsf{M}} + s\,\mathsf{D}^{\mathsf{M}}\right)\,\underline{\tilde{\varphi}}, \tag{2.11}$$

where the superscript M indicates the $[M \times N]$ matrices collecting the integral coefficients which relates the value of $\tilde{\varphi}$ and $\tilde{\chi}$ on the boundary, with the solution at the microphones.

2.1.3 CHIEF regularization

The use of a the KHIE for the analysis of the radiation and scattering of a closed surface $\mathcal{S}$ within the free space is affected by the so–called *fictitious eigenfrequencies* problem. In few words, the scattered acoustic field in the unbounded domain surrounding $\mathcal{S}$, exhibits non–physical resonances, which are related to the eigensolutions of the interior problem defined in the (fictitious) acoustic cavity enclosed by $\mathcal{S}$. Specifically, the solution of the exterior problem with Neumann boundary conditions is singular at the eigenfrequncies of the interior Dirichlet problem, whereas the eigenfrequencies of the interior Neumann BVP affects the solution of the exterior Dirichlet problem (see *e.g.*, Colton & Kress, [6]). AcouSTO includes the possibility to regularize the BEM exterior solution using the CHIEF technique (Combined Helmholtz Integral Equation Formulation, Schenck [3]). The advantage of this technique resides its implementation simplicity; the drawback is a reduced reliability at high frequency or, in general, at those frequencies corresponding to a range of high modal density for the enclosure bounded by $\mathcal{S}$.[2]

The basis of the CHIEF technique is the introduction of N^{C} additional collocation points inside $\mathcal{S}$. At these additional locations, the solution of the exterior problem is zero, and the system of equations 2.8 can be augmented by the additional equations corresponding

[2]The use of a more reliable regularization technique, such as the CONDOR (Composite Normal Derivative Overlapping Relation) method [11], is certainly one of the improvement that will be included in one of the future releases of AcouSTO .

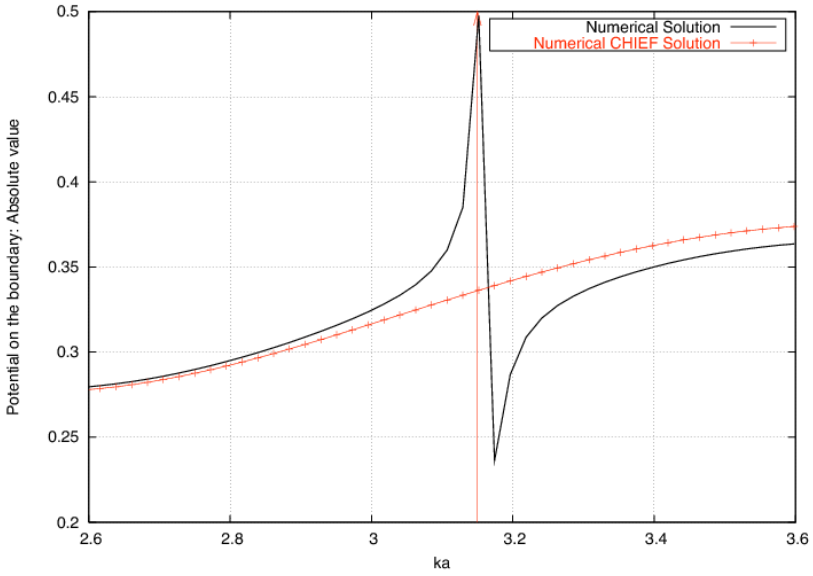


Figure 2.1: CHIEF regularization of the acoustic potential on the boundary of a unit sphere. The amplitude spectrum in the vicinity of the fictitious eigenfrequency corresponding to a reduced frequency $ka = \pi$ is compared with the un-regularized solution.

to the CHIEF points. This yields an over–determined system of $N + N^{\mathsf{C}}$ equations for N unknown. The solution can be found by minimizing the quadratic form

$$\mathsf{J}(\underline{\tilde{\varphi}}) = \frac{1}{2} \left(\hat{\mathsf{Y}}\, \underline{\tilde{\varphi}} - \underline{\tilde{\mathsf{b}}} \right)^{*T} \left(\hat{\mathsf{Y}}\, \underline{\tilde{\varphi}} - \underline{\tilde{\mathsf{b}}} \right) \tag{2.12}$$

where

$$\hat{\mathsf{Y}} = \begin{bmatrix} \mathsf{Y} \\ -\mathsf{C}^{\mathsf{C}} - s\,\mathsf{D}^{\mathsf{C}} \end{bmatrix}, \qquad \underline{\tilde{\mathsf{b}}} = \begin{bmatrix} \mathsf{B} \\ \mathsf{B}^{\mathsf{C}} \end{bmatrix} \underline{\tilde{\chi}}, \tag{2.13}$$

and the superscript C indicates the coefficients influencing the CHIEF collocation points. The solution is given by

$$\underline{\tilde{\varphi}} = \left(\hat{\mathsf{Y}}^{*T}\, \hat{\mathsf{Y}} \right)^{-1} \hat{\mathsf{Y}}^{*T}\, \underline{\tilde{\mathsf{b}}}. \tag{2.14}$$

Additional details will be given in the advanced user's and programmer's manual, or can be found in Gennaretti & Iemma [5] or in Seybert & Rengarajan [4].

For a basic use of AcouSTO the user has only to define the locations of the internal collocation points, following the procedure explained in Section 4.1. Figure 2.1 shows the effect of the CHIEF regularization technique on the spectrum

2.2 What AcouSTO can do

Using the formulation presented in the previous Section, the following classes of problems can be solved:

- scattering of planar or spherical waves by multiple, arbitrarily shaped, bodies ;
- radiation of vibrating closed surfaces with wall motion assigned.

There is no limitation in the number of incident waves. Moreover, the possibility to include an arbitrary incident field from external data (let's say, the pressure field calculated with a CFD code or from experiments) is in preparation. This option will be available in the next release of the code.

Both interior and exterior problem can be solved, providing the appropriate topology in the definition of the BEM mesh (see Section 3.1).

The present version of the code **can't** handle moving surfaces, nor moving sources. These features will be included soon.

NOTE
With AcouSTO one can choose two different approaches to the solution: in the first one (formulation A), Eq. 2.5 is written for the total field, which is the sum of the incident and the scattered fields, $\tilde{\varphi} = \tilde{\varphi}_{inc} + \tilde{\varphi}_{sc}$, whereas in the second (formulation B), the unknown field is $\tilde{\varphi}_{sc}$, and the effect of the incident field is included in the boundary conditions. The choice between the two formulations can be made following the instructions presented in Section 4.2

CHAPTER 3

Setup of domain geometry

This is where you learn about the setup of geometries, meshes and microphones

The structure of the input files needed by AcouSTO is explained in detail in Appendix A. Basically, the user has to prepare a main configuration file plus a bunch of additional files. The number and the content of these files depend on the problem to be solved. In the present chapter, the fundamental rules required to properly setup our acoustic BEM simulation are given. Specifically, the BEM mesh topology is described in Section 3.1, whereas the symmetry management is addressed in Section 3.3.

Starting from AcouSTO v1.6, the possibility to import a boundary mesh generated with the public domain grid generator Gmsh (http://geuz.org/gmsh/) has been included. AcouSTO is compatible with Gmsh version $\geq$ 2.7. Only ASCII .msh files are accepted, using format version $\geq$ 2.2. Details about the Gmsh interface are given is Section 3.4.

It is important to notice that the management of symmetries when using Gmsh files as an input slightly differs from the one used with the native AcouSTO format. Details are given in Section 3.5.

3.1 BEM grid topology

AcouSTO can only generate simple geometries (sphere, cylinder, plate). For more complex geometries you must provide a "nodes" file following the syntax described in Section B. You can generate the geometry using the powerful modeler Blender (http://www.blender.org) and export the AcouSTO nodes file with the plugin provided with the distribution (see Appendix C). Alternatively, you can feed AcouSTO with a mesh file produced with Gmsh (http://gmsh.info). Although all methods are perfectly equivalent, we strongly suggest to adopt Gmsh as the standard modeling tool.

Whatevert method you use, yopu must have a clear idea of the way AcouSTO handles the BEM grid. Always pay attention to the ordering of the panels' vertices, for it defines the orientation of the panel surface, *i.e.*, the orientation of the vector normals to each element. The differential description of the panel geometry depends on this choice, and a wrong order of the vertices results in a miscalculation of the local covariant base vectors. The local curvilinear coordinates, ξ_1 and ξ_2, are such that $\xi_k \in [-1, 1]$, for $k = 1, 2$ (see Figure 3.1). The covariant base vectors and the unit normal to the panel are defined as

$$\mathbf{a}_1 = \frac{\partial \mathbf{x}}{\partial \xi_1}, \quad \mathbf{a}_2 = \frac{\partial \mathbf{x}}{\partial \xi_2}, \quad \hat{\mathbf{n}} = \frac{\mathbf{a}_1 \times \mathbf{a}_2}{|\mathbf{a}_1 \times \mathbf{a}_2|}. \tag{3.1}$$

This vectors are evaluated numerically by averaging the finite differences along the panel edges. Indicating with $\vec{v}_i$, $i = 1, ..., 4$ the position vector of the panel vertices, yields

$$\mathbf{a}_1 \simeq \frac{1}{4}(\mathbf{v}_1 - \mathbf{v}_4 + \mathbf{v}_2 - \mathbf{v}_3) \tag{3.2}$$

$$\mathbf{a}_2 \simeq \frac{1}{4}(\mathbf{v}_2 - \mathbf{v}_1 + \mathbf{v}_3 - \mathbf{v}_4). \tag{3.3}$$

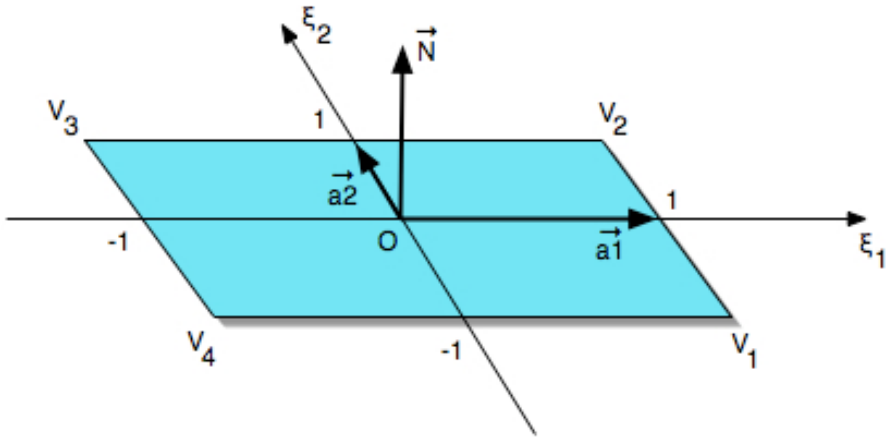


Figure 3.1: Local curvilinear coordinates and covariant base vectors on the panel.

The approximate base vectors obtained with Eq. 3.2 are used to evaluate the unit normal to the panel according to Eq. 3.1. Clearly, the order of the nodes defining the sequence of the panel vertices influences the orientation of the unit normal (see Fig. 3.2).

The unit normal vector points towards the side of the panel from which the sequence of the vertices is seen as counterclockwise.

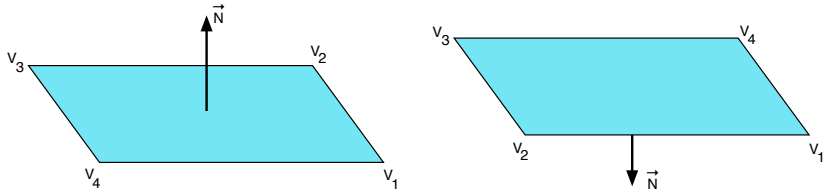


Figure 3.2: Unit normal orientation.

3.2 AcouSTO native nodes file format

The AcouSTO native format of node files is very simple. The first part of the file is simply the list of the x, y, and z coordinates of all the nodes, separated by spaces. The second part of the file contains the definition of the panels as blocks of four integer number corresponding to the indices of the nodes that are vertices of the panels. If, for example, the vertices of the k^{th} panel are the i^{th}, j^{th}, n^{th}, m^{th} nodes, then the k^{th} line of the second block of the file will be `i j n m`. The indices are stored in the array `jnodb(NVERT)`, which

is the topology connection function between the panel vertices numbering system and the nodes numbering system.

Although the integration subroutines used in AcouSTO can handle triangular panels, the topology connection function used to define the local curvilinear coordinates system supports only four vertices per panel. The only way to include triangular panels is to locate two adjacent vertices on the same point.

Figure 3.3 depicts a very simple example. Let's assume a cubic surface divided into six panels, corresponding to its six faces. The eight nodes N_i, $i = 1, ..., 8$, are ordered as in figure. Assume also that node N_5 corresponds to the origin of the global coordinates system used, and that the cube edges are 1 m long. This mesh (useless, for practical purposes) can be used to analyze both the interior and exterior problems, depending on the orientation of the unit normal vectors. As we have seen, the latter depends on the

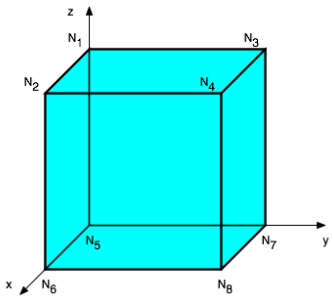


Figure 3.3: A simple example.

ordering of the vertices. In the following, the structure of the geometry file is given for the outward and inward pointing normals, respectively.

Mesh 1 - Content of node file for test case in Fig. 3.3 with normal vectors pointing **outward**.

```
0.0  0.0  1.0
1.0  0.0  1.0
0.0  1.0  1.0
1.0  1.0  1.0
0.0  0.0  0.0
0.0  1.0  0.0
1.0  1.0  0.0
```

```
7   5   1   3
6   8   4   2
6   2   1   5
8   7   3   4
5   7   8   6
4   3   1   2
```

Mesh 2 - Content of node file for test case in Fig. 3.3 with normal vectors pointing **inward**.

```
0.0  0.0  1.0
1.0  0.0  1.0
0.0  1.0  1.0
1.0  1.0  1.0
0.0  0.0  0.0
0.0  1.0  0.0
1.0  1.0  0.0
7   3   1   5
6   2   4   8
6   5   1   2
8   4   3   7
5   6   8   7
4   2   1   3
```

3.3 Symmetry management in AcouSTO native format

The code AcouSTO can take advantage of the symmetry of the problem to reduce the numerical effort required for the solution. If a symmetry plane is present, the collocation points used to evaluate the solution are distributed only along one of the two symmetric halves of the boundary surface, thus dividing by a factor two the number of equations of the resulting BEM system. Similarly, in presence of axial symmetry, one can evaluate the solution only along one half of a meridian circle, and the reduction of the computational effort depends on the number of "slices" in which the axially-symmetric boundary has been divided. AcouSTO can take into account symmetry through the value of the integer variable KSYMMI. This variable controls the ratio between the number of collocation points NCNTR and the number of elements NELMB according to the following criterion:

$$\frac{\texttt{NELMB}}{\texttt{NCNTR}} = 2^{(\texttt{KSYMMI})} \qquad \text{for } \texttt{KSYMMI} \leq 3$$

$$\frac{\texttt{NELMB}}{\texttt{NCNTR}} = \texttt{KSYMMI} \qquad \text{for } \texttt{KSYMMI} > 3.$$

In summary, KSYMMI ≤ 3 is associated to the existence of symmetry planes, whereas KSYMMI > 3 indicates axialsymmetry (see Table 3.3).

KSYMMI	kind of symmetry	NCNTR/NELMB
0	no symmetries	1
1	one symmetry plane	0.5
2	two symmetry planes	0.25
3	three symmetry planes	0.125
> 3	axial symmetry	$1/N$

Table 3.1: Relationship between KSYMMI and the ratio NCNTR/NELMB.

The last column of Table 3.3 reports the ratio between the number of collocation points NCNTR and the number of elements NELMB, in order to give an idea of the reduction of the computational effort resulting from the use of the symmetry options. Note that the dimensions of the matrices involved in the final system of equations is NCNTR×NCNTR, and that the number of integral coefficients that have to be evaluated to build-up these matrices is equal to $3 \times$ NCNTR $\times$ NELMB.

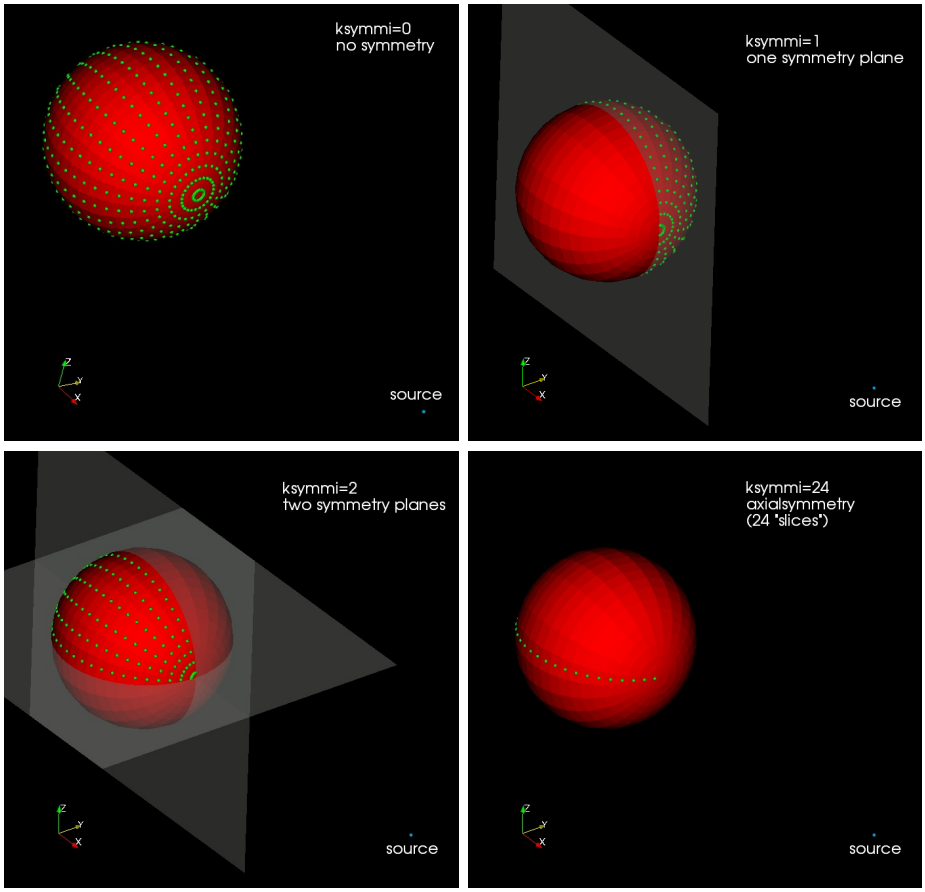


Figure 3.4: Control points distributions (green points) for KSYMMI=0, 1, 2, 24 (from top/left to bottom/right). KSYMMI=1 and 2 are possible because of the even number of slices used.

The present version of the code does not have a dedicated geometry pre-processor, so, particular care must be taken in the construction of the BEM mesh and its topology when the symmetry option is enabled. The code reads the nodes coordinates and panel topology from the geometry file, according to the criteria described in Sections 3.1 and B.1, and simply builds NCNTR collocation points as the centroids of the first NCNTR elements. So, in the definition of the BEM grid topology, the user must provide the boundary elements following the instructions in Table 3.3

KSYMMI	**BEM grid structure**
0	No restrictions.
1	the second half of the elements must be obtained by mirroring the first half with respect to the symmetry plane.
2	the second quarter of elements must be obtained by mirroring the first quarter w.r.t. the first symmetry plane **and** the second half of elements must be obtained by mirroring the first half w.r.t. the second symmetry plane
3	the second eighth of elements must be obtained by mirroring the first eighth w.r.t. the first symmetry plane **and** the second quarter of elements must be obtained by mirroring the first quarter w.r.t. the second symmetry plane **and** the second half of elements must be obtained by mirroring the first half w.r.t. the third symmetry plane
> 3	the k^{th} "slice" of the surface must be obtained by rotating the first NELMB/KSYMMI elements (the first "slice") by $(k-1)2\pi/$KSYMMI radiants.

Table 3.2: BEM grid constraints in presence of symmetry.

Figure 3.4 shows the geometries generated to solve a symmetric problem: the scattering of the field generated by a monopole source located at $(5, 0, 0)$ by a sphere, discretized with a parallels/meridians mesh with north and south poles along the x–axis. The number of meridians sectors is equal to 24. Such a problem presents intrinsic axial symmetry around the x–axis, but the specific way the BEM mesh was built allows the user to solve the problem exploiting only the symmetry with respect to the xy-plane and/or xz-plane. The geometry obtained with the different approaches are depicted in Fig. 3.4.

3.4 Gmsh input files

Version 1.6 of AcouSTO introduces the possibility to read meshes generated with the public domain grid generator Gmsh (http://geuz.org/gmsh/), developed by Christophe

Geuzaine and Jean-Francois Remacle. Gmsh is an easy–to–use yet powerful grid generator with built–in CAD and post–processing capabilities. Although Gmsh has its own format to export the mesh files (the one that must be used as an input in AcouSTO), it is compatible with most of the standard public formats. A detailed documentation, including tutorials and screencasts can be found on the Gmsh site (http://geuz.org/gmsh/. Here we limit the description of the use of Gmsh to the specific aim of generating a mesh compatible with AcouSTO .

First, it must be emphasized that the AcouSTO interface to Gmsh is compatible only with file format version 2.0, and it has been tested only with Gmsh versions ≥ 2.7. Figure 3.5 shows a screenshot of the Gmsh GUI. The CAD capabilities built–in the program are sufficient to

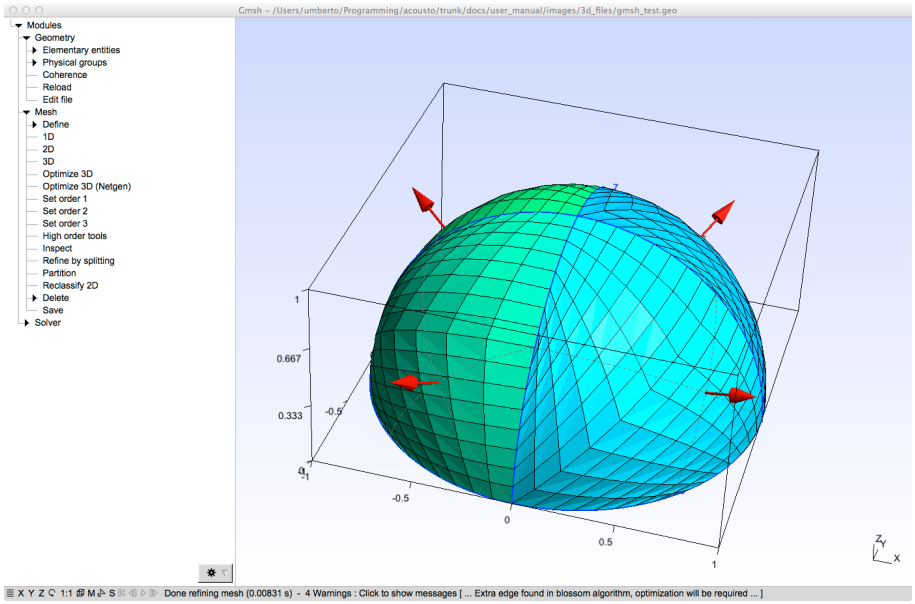


Figure 3.5: A screenshot of the Gmsh GUI (v2.8 on MacOSX 10.7.5).

easily generate geometries based on simple primitives using only the GUI. Nevertheless, the simple syntax of the .geo files and the availability of versatile spline primitives allow for a straightforward generation of more complex geometry, simply by producing the .geo file using direct programming in a standard language, like C, Fortran or whatever you like. Figure 3.6 depicts the complicated geometry of an unconventional aircraft developed at the *Roma Tre* University. The points along the sections of wing and fuselage, as well as along the profile of leading and trailing edges were produced with an external, in–house written, Fortran code and exported in an ASCII .geo file. The Gmsh spline primitive was used to build the lines through these points. The surface and the mesh were then generated in Gmsh (Figs. 3.6 and 3.7) and imported in AcouSTO to estimate the effect of such a configuration on the scattering of the engine noise components (See Fig. 3.8).

In the next subsection, the procedure to obtain with Gmsh a surface grid compatible with AcouSTO is described. Note that we will address only the basic process, without entering

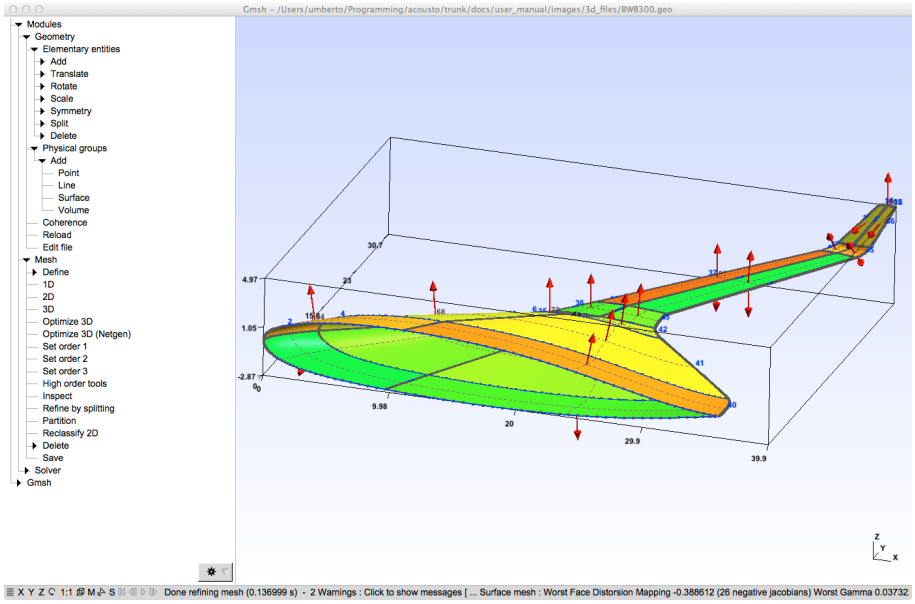


Figure 3.6: Gmsh screenshot of a complex, unconventional aircraft geometry generated by a simple fortran code using three–dimensional Hermite interpolation of the aircraft sections.

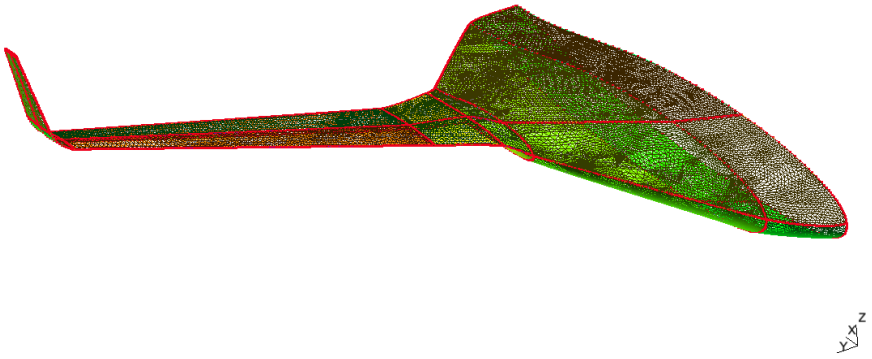


Figure 3.7: Example of the mesh generated with Gmsh on the geometry depicted in Fig. 3.6.

into details for what concern the fine tuning of the local or global properties of the grid. Gmsh has a lot of possibility to control in detail the quality of the mesh, described clearly in the documentation available on the developers' web site.

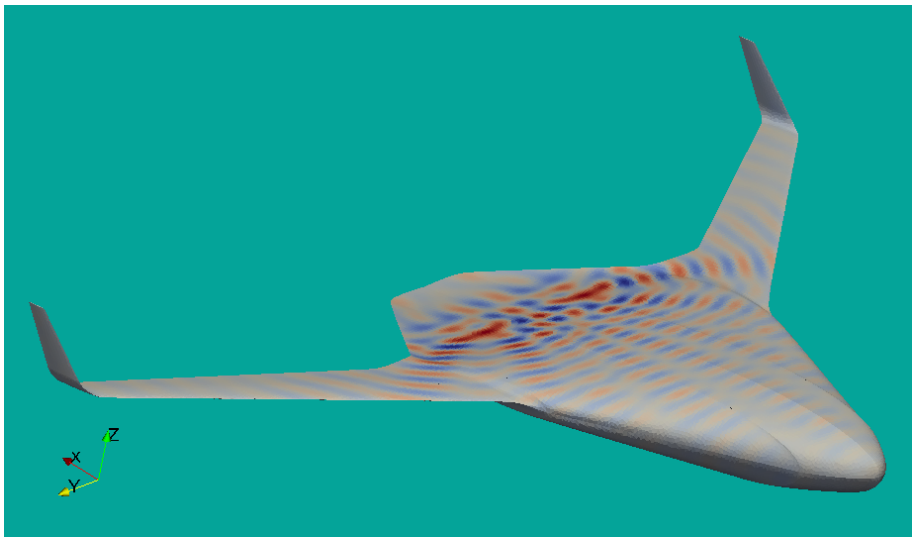
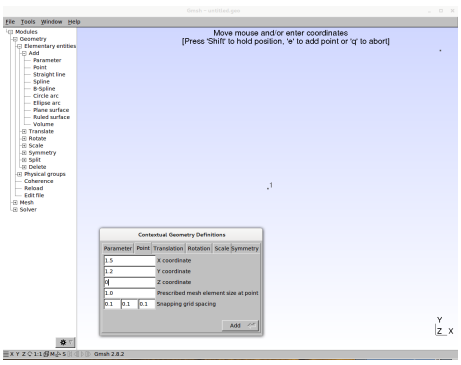


Figure 3.8: Example of the AcouSTO result obtained by mirroring the mesh of Fig. 3.7 w.r.t the xz plane (see Section 3.5).

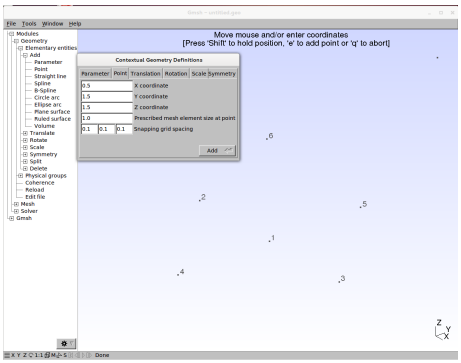
3.4.1 Building a mesh for AcouSTO in Gmsh

Let's assume that we are interested in the scattering of a sphere within an acoustic environment having one plane of symmetry passing through the center of the sphere. According to the AcouSTO management of symmetries with the Gmsh file format (see next Section) we have to build in Gmsh only a hemisphere. The steps are listed below.

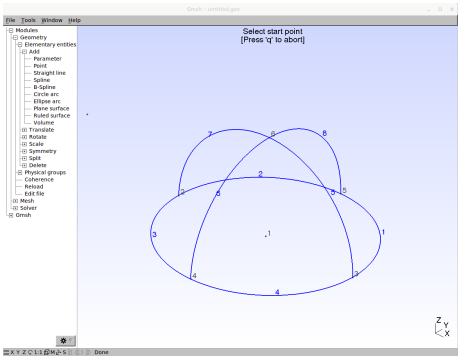
Step 1 - First, add the point where the center of the sphere is located, using the appropriate entry of the menu on the left.



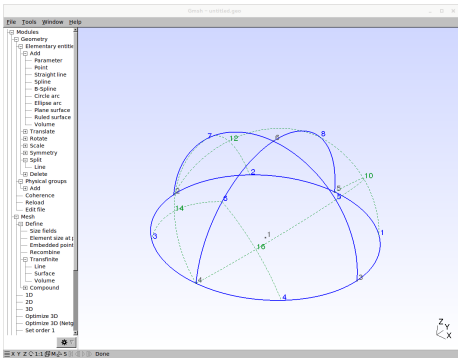
Step 2 - Then, using the same tool, add the five end points of three orthogonal diameters of the sphere parallel to axes of the global frame of reference.



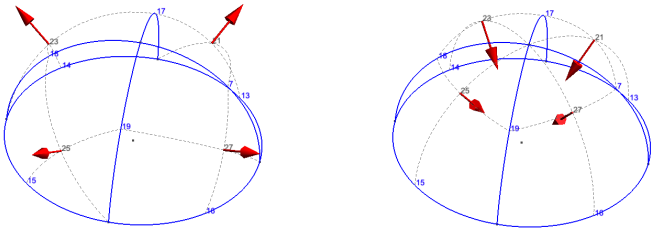
Step 3 - Select the `circle arc` primitive in the `Add` menu and build the eight arcs as in the figure below. Note that the arcs must be eight to build properly the sectors of the sphere.



Step 4 - Select the `ruled surface` primitive from the `Add` menu and build the four surfaces as in the figure below.



Step 5 - Visualize the normals to the generated surfaces and verify their orientation. This is very important because it defines the domain of your acoustic simulation. The figures below show the case of outward (left) in inward (right) oriented normals. `AcouSTO` accepts both the orientation. In the first case, the acoustic domain will be the exterior unbounded field, whereas in the second, the interior spherical cavity.



Most likely, not all the surfaces will have the same orientation (it depends on how the boundaries were built). The easiest way to change the orientation is by editing the `.geo` and modify the sign of the surfaces you want to flip in the appropriate section of the file, as in the examples below

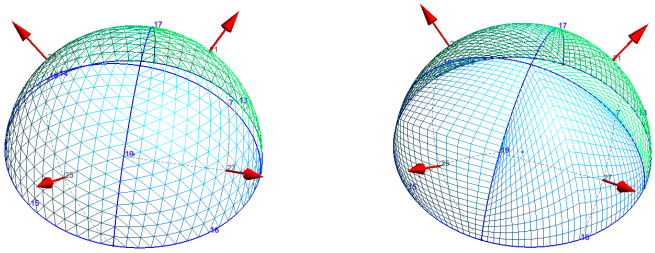
```
...
Line Loop(20) = {13, 17, 7};
Ruled Surface(21) = {20};
Line Loop(22) = {17, -14, -18};
Ruled Surface(23) = {-22};
Line Loop(24) = {18, -19, -15};
Ruled Surface(25) = {-24};
Line Loop(26) = {19, 7, -16};
Ruled Surface(27) = {-26};
```

Outward normals

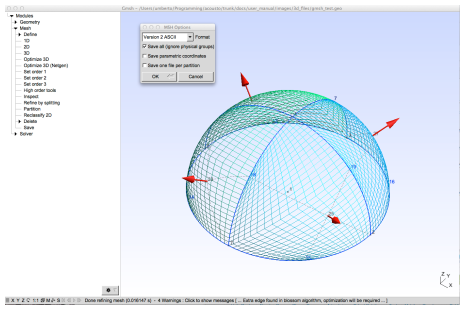
```
...
Line Loop(20) = {13, 17, 7};
Ruled Surface(21) = {-20};
Line Loop(22) = {17, -14, -18};
Ruled Surface(23) = {22};
Line Loop(24) = {18, -19, -15};
Ruled Surface(25) = {24};
Line Loop(26) = {19, 7, -16};
Ruled Surface(27) = {26};
```

Inward normals

Step 6 - Once that you have all the normals properly oriented, you can generate the mesh by first clicking the `1D` entry of the `Mesh` menu and then the `2D` entry. The mesh can be refined by splitting, simply clicking on the dedicated menu item as many times as you want.



Step 7 - When you are happy with your mesh, select `Save as...` from the `File` menu and choose the `.msh` format. In the dialog that appears after the filename and location selection you must choose `Version 2 ASCII` as format, and (just to be sure, but this is not mandatory) check the `Save all` check–box.



Now, you are ready to import your Gmsh grid in AcouSTO , follow it the instructions given in Section 3.5.

The above procedure was described referring to the Gmsh GUI menus for the sake of simplicity. Nevertheless, after a brief training with Gmsh you will probably prefer to edit directly the .geo ASCII file. This is actually the way we used to produce all the meshes we are using.

It must be pointed out that Gmsh and AcouSTO use a different numbering system for the panels vertices, as well as a slightly different convention for the local coordinates. Figure 3.9 shows these differences for triangular and quadrilateral panels (the only accepted by AcouSTO). This aspect is not crucial to run the code, but could be of interest to people interested in modifying the sources. The only thing that must be kept in mind is that AcouSTO uses a 4–vertices topology (with 2 vertices coincident) also for triangular panels.

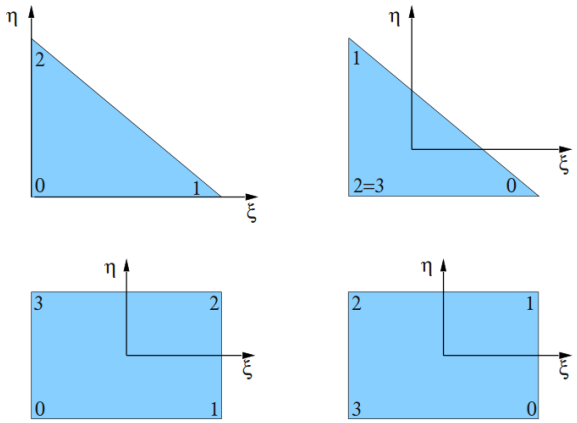


Figure 3.9: Gmsh (left) and AcouSTO (right) panel vertices numbering and local coordinates.

3.5 Symmetry management in Gmsh mesh format

Of course, AcouSTO can take advantage of the symmetry of the problem also in the case of meshes generated by Gmsh . Nevertheless, the possibility to work with unstructured meshes, generated by the external software using specific partitioning algorithms of the manifold, makes impossible the approach adopted for meshes in native format (see Section 3.3). For this reason, a different approach to symmetry is used with Gmsh files. The fundamental differences are:

- The user must provide to AcouSTO only the *un–symmetrized* portion of the domain boundary AcouSTO will generate the complete *symmetrized* geometry according to the symmetrization directives specified in the config file.
- With Gmsh files, the domain can have only one plane of symmetry or has to be axially symmetric.

As for the AcouSTO native format, the parameter controlling the existence of symmetries is KSYMMI available in the runinfo module. The symmetrization directives are specified giving in input the vector symvec and the reference point refpoint in a block of type gmsh (see Section A.2). Their meaning deepens on the context, and is explained in Table 3.5.

KSYMMI	NELMB/NCNTR	**Meaning of input variables**
0	1	not active
1	2	symvec is the vector orthogonal to the symmetry plane passing through refpoint (see Figures 3.10)
2	not active	program stops after an error message
3	not active	program stops after an error message
≥ 4	KSYMMY	the vector symvec applied on refpoint fixes the symmetry axis (see Figures 3.13)

Table 3.3: Symmetry management with Gmsh mesh files

The relevant sections of the config files related to the examples depicted in Figures **??** and 3.13 are given in Table 3.4.

Note that in presence of multiple bodies the use of symmetries makes the format of the .vtk and .msh fail, but not the solution. So, for your. visualization you must use the .out files

Table 3.4: Relevant sections of the config files for the two examples used.

Wing	**Fan + nacelle**
...	...
runinfo={	runinfo={
active = 1;	active = 1;
title = "Wing_Sym";	title = "Nacelle_Axial";
owner = "Umberto Iemma";	owner = "Umberto Iemma";
ksymmi=1;	ksymmi=24;
krow =-1;	krow =-1;
vsound =340.0;	vsound =340.0;
};	};
wing={	nacelle={
type="gmsh";	type="gmsh";
filename="wing.msh";	filename="fan_nacelle.msh";
symvec={x=0.;y=1.;z=0.;};	symvec={x=0.;y=0.;z=1.;};
refpoint={x=0.;y=0.;z=-1.;};	refpoint={x=0.;y=0.;z=0.;};
};	};

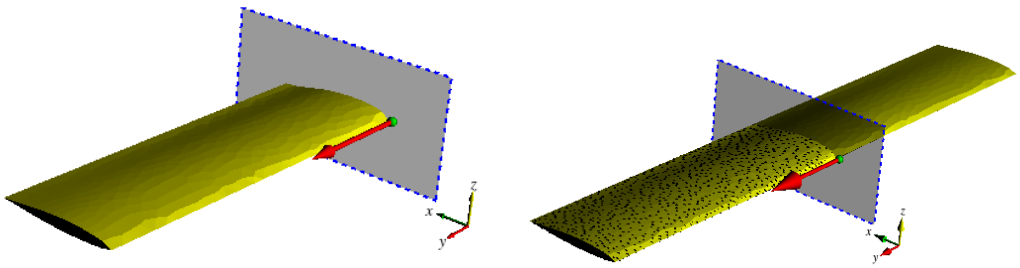


Figure 3.10: Un-symmetrized mesh of the wing–like geometry as generated in Gmsh (left) and complete geometry after AcouSTO symmetrization. The vector symvec (red) is orthogonal to the symmetry plane (gray) passing through refpoint (green). Collocation points (black dots) appear only on the original portion of the geometry.

3.6 Field microphones in Gmsh mesh format

Starting from AcouSTO v1.6, the field microphones can be provided in form of Gmsh .msh format. This possibility is strongly encouraged, when the solution on a 2D manifold [1] in the field is needed. Indeed, the use of Gmsh allows for the generation of much more

[1] The possibility to distribute the microphones on a 3D subdomain is not available yet and wil be included in a future release.

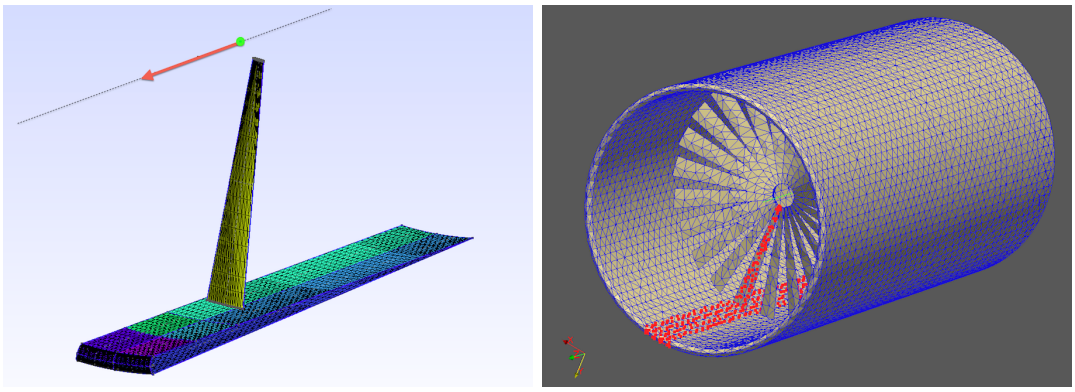


Figure 3.11: Gmsh mesh of one fan blade and a sector of nacelle corresponding to $\pi/12$ radians (left) and the complete mesh of the fan–nacelle geometry after AcouSTO axial symmetrization. The vector symvec (red) through refpoint (green) fixes the axis of the nacelle. Collocation points (red dots) appear only on the original portion of the geometry.

complex microphone geometries in a simple reliable way. In addition, the post-processing of the solution in the field yields much better results, thanks to the structured topology of the field monitoring points (see Section 4.8). The procedure to generate a microphone input file in Gmsh is very simple.

Step 1 - First generate in Gmsh the surfaces where you want to distribute microphones.

Step 2 - Generate a physical group with all these surfaces (optionally, you can edit the .geo file to associate a custom tag to it), and mesh it using the standard procedure.

Step 3 - Export the mesh, leaving unchecked the option Save all (ignore physical groups).

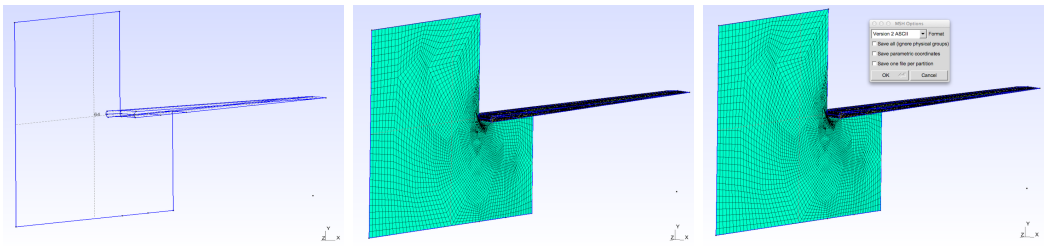


Figure 3.12: Building a Gmsh microphones mesh (Steps 1-3, left to right).

If you provide the file so obtained to AcouSTO , it recognize automatically the Gmsh format and reads mics and topology. Note the with a Gmsh microphone file **you don't need to provide** *NMICS* **in the config file** (it is read from the file). Figure **??** shows the solution visualization on the Gmsh microphones mesh using a VTK viewer and the post–processing features of Gmsh itself.

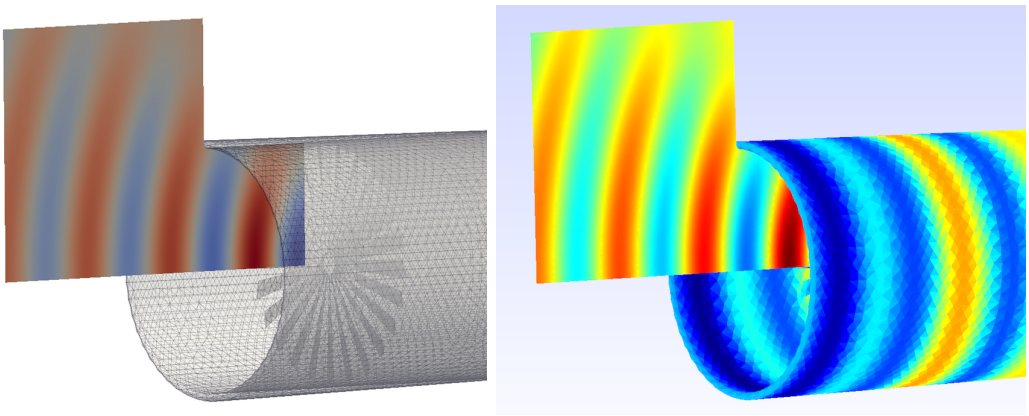


Figure 3.13: Visualization of solution on a Gmsh microphone mesh with a VTK viewer (left) and with Gmsh .

CHAPTER 4

Analysis setup, solution and post-processing

This is where you can finally get some number

In this chapter we outline the steps needed to setup the code solver, as well as the fundamental post-processing techniques available to visualize the results. One of the key points is the proper definition of the boundary conditions associated with the problem. So, we will start with Section 4.2, where an accurate (even if not exhaustive) description of the theoretical formulation used to treat this specific issue is presented. Then, in Section 4.1 the definition of the acoustic properties of the boundary is outlined. Section 4.4 deals with the proper setup of the memory management variable, extremely useful to identify a trade-off between memory allocation and access to disk. Finally, Section 4.8 describes how to analyze the produced results using only freely available tools.

4.1 The incident field

With "acoustic environment" we mean all the parameters related to the acoustic properties of the boundary (wall impedance and radiating panels), the elementary sources present in the domain, and the primary, or *incident*, acoustic field. The setup of these properties can be implemented through dedicated variables in the configuration file, and the corresponding data files. In this section we focus the attention on the definition of the sources and the incident field. The treatment of the boundary conditions deserves a more detailed discussion which will be presented in the following sections.

The current version od `AcouSTO` can handle only isolated monopoles and incoming plane waves. This limitation will be removed soon, including the possibility to consider isolated dipoles and quadrupoles, as well as incident fields calculated by external tools and provided in form of data files.

For the moment, there here are no limits in the number of monopoles and/or incoming plane waves. The relevant variables are presented in Tables 4.1 and 4.2.

Table 4.1: **Point sources (monopoles)**

variable	config block	meaning
`nsourc`	`modsol`	number of monopoles in the field
`sources_file`	`modsol`	name of the file with the locations and intensities of the monopoles

When one of the above dimensions is not zero, the corresponding input files must be provided using the format specifications given in Section B.2. The content of each of the above files is ignored when the corresponding dimension is set to zero.

`AcouSTO` can handle two different formulations of the problem. In the first one (**formulation A** in the following) the solution is found for the total field $\tilde{\varphi}$. It is convenient when acoustic sources are present in the field, provided that the associated primary field could be easily

Table 4.2: **Plane waves**

variable	config block	meaning
`nplanw`	`modsol`	number of incoming plane waves
`planw_file`	`modsol`	name of the file containing the wave vector of the incoming waves

calculated (see Section 4.2). In such a case, the integral form of the problem is derived directly from the non–homogeneous wave equation, and is written in terms of the total acoustic perturbation $\tilde{\varphi}(\mathbf{x}, \kappa)$ (formulation A). In the second approach (**formulation B**) the problem is solved by calculating the scattering component of the acoustic field, $\tilde{\varphi}_{\text{sc}}$. This is applicable when the primary field is directly known (*e.g.*, from an impinging wave or from external calculations). The equation is recast in form of homogeneous wave equation for the scattered field, and the effect of the incident field is given through the boundary conditions.

4.2 General formulation for the boundary conditions

The boundary conditions associated to the wave propagation within the domain $\mathcal{V}$ can be derived from the relationship existing on the boundary between the unknown function and its normal derivative, *i.e.*,

$$\gamma(\mathbf{x}, \kappa)\, \tilde{\varphi}(\mathbf{x}, \kappa) + \lambda(\mathbf{x}, \kappa)\, \frac{\partial \tilde{\varphi}(\mathbf{x}, \kappa)}{\partial n} = \tilde{f}(\mathbf{x}, \kappa) + \tilde{\mathbf{g}}(\kappa) \cdot \mathbf{n}(\mathbf{x}), \qquad \text{for } \mathbf{x} \in \partial\mathcal{V} \tag{4.1}$$

where γ, λ, and f are known complex functions of position and frequency, whereas $\tilde{\mathbf{g}}$ is a vector constant in space (possibly varying only with frequency). *Dirichlet–*, *Neumann–*, or *Robin–* type boundary conditions can be easily derived from Eq. 4.1 giving the appropriate form to γ, λ, f and $\mathbf{g}$. The discrete form of Eq. 4.1 is the following

$$\mathsf{\Gamma}\, \underline{\tilde{\varphi}} + \mathsf{\Lambda}\, \underline{\tilde{\chi}} = \underline{\tilde{\mathsf{f}}} + \underline{\tilde{\mathsf{a}}} \tag{4.2}$$

where $\underline{\tilde{\mathsf{f}}}$ and $\underline{\tilde{\mathsf{a}}}$ are the column vectors collecting the values of $\tilde{f}$ and $\tilde{\mathbf{g}} \cdot \mathbf{n}$ at the collocation points, whereas $\mathsf{\Gamma}$ and $\mathsf{\Lambda}$ are diagonal matrices collecting the values of the γ and λ functions at the collocation points

$$\mathsf{\Gamma} = \begin{bmatrix} \gamma(\mathbf{x}_1) & 0 & 0 & 0 \\ 0 & \gamma(\mathbf{x}_2) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \gamma(\mathbf{x}_N) \end{bmatrix}, \quad \mathsf{\Lambda} = \begin{bmatrix} \lambda(\mathbf{x}_1) & 0 & 0 & 0 \\ 0 & \lambda(\mathbf{x}_2) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda(\mathbf{x}_N) \end{bmatrix} \tag{4.3}$$

It is important to clarify what do we mean in the following with *Neumann–* and *Dirichlet–type* boundary conditions.
When a linear combination of $\tilde{\varphi}$ and $\tilde{\chi}$ is known on $\partial\mathcal{V}$ (*i.e.*, when both λ and γ are different from zero) we are in presence of *Robin* boundary conditions. In the following, we define Neumann–*type* boundary conditions the expression obtained using Eq. 4.1 to write $\tilde{\chi}$ as a function of $\tilde{\varphi}$. Conversely, when Eq. 4.1 is used to express $\tilde{\varphi}$ as a function of $\tilde{\chi}$ we talk about Dirichlet–*type* conditions.
This is only a naming convention introduced to distinguish between the two different numerical approaches used and has nothing to do with the rigorous definition of Dirichlet conditions ($\lambda \equiv 0$) and Neumann conditions ($\gamma \equiv 0$).

4.2.1 *Neumann–type* boundary conditions

As pointed out at the end of the previous section, we talk about Neumann–type boundary conditions when Eq. 4.1 is used to express $\partial\tilde{\varphi}(\mathbf{x},\kappa)/\partial n$ as a function of $\tilde{\varphi}$. If $\gamma = 0$ this formulation yields pure Neumann conditions (*i.e.*, normal derivative known on the boundary). In such a case, Eq. 4.2 is rewritten as

$$\underline{\tilde{\chi}} = \mathsf{\Lambda}^{-1}\left(\underline{\tilde{\mathsf{f}}} + \underline{\tilde{\mathsf{a}}}\right) - \mathsf{\Lambda}^{-1}\mathsf{\Gamma}\,\underline{\tilde{\varphi}} \tag{4.4}$$

The resulting solving system of equations for $\underline{\tilde{\varphi}}$ for the two formulation is derived in the following.

Formulation A

In this first approach, the integral formulation is written in terms of total acoustic perturbation $\tilde{\varphi}(\mathbf{x},\kappa)$, and derived from the non-homogeneous wave equation 2.1, to yield

$$E(\mathbf{y})\,\tilde{\varphi}(\mathbf{y}) = \oint_{\mathcal{S}} \left(G_0 \frac{\partial\tilde{\varphi}}{\partial n} - \tilde{\varphi}\frac{\partial G_0}{\partial n} + s\,\tilde{\varphi} G_0 \frac{\partial\theta}{\partial n}\right) e^{-s\,\theta}\, d\mathcal{S}(\mathbf{x}) + \mathcal{I}_{\mathcal{V}}(\mathbf{y},\kappa) \tag{4.5}$$

where

$$\mathcal{I}_{\mathcal{V}}(\mathbf{y},\kappa) = \int_{\mathcal{V}} G_0\,\tilde{q}\,e^{-s\,\theta}\, d\mathcal{V}(\mathbf{x}). \tag{4.6}$$

and with the boundary conditions given by Eq. 4.1. In general, the volume integral arising from the presence of the acoustic forcing term cannot be calculated in closed form, and its numerical treatment is not possible. On the other hand, for certain types of acoustic

sources, the value of $\mathcal{I}_{\mathcal{V}}(\mathbf{y}, \kappa)$ can be calculated exactly (see Section 4.2.1), and the matrix form of the BEM system is

$$\frac{1}{2}\underline{\tilde{\varphi}} = \mathsf{B}\,\underline{\tilde{\chi}} + (\mathsf{C} + s\,\mathsf{D})\,\underline{\tilde{\varphi}} + \underline{\tilde{\mathsf{q}}} \tag{4.7}$$

where $\underline{\tilde{\mathsf{q}}}$ is the $[M \times 1]$ column vector collecting the value of the acoustic field induced by the source at the collocation points, and $\underline{\tilde{\chi}}$ can be derived from Eq. 4.4 (see Section 4.1), to yied

$$\frac{1}{2}\underline{\tilde{\varphi}} = \mathsf{B}\,\left[\Lambda^{-1}\,\left(\underline{\tilde{\mathsf{f}}} + \underline{\tilde{\mathsf{a}}}\right) - \Lambda^{-1}\Gamma\,\underline{\tilde{\varphi}}\right] + (\mathsf{C} + s\,\mathsf{D})\,\underline{\tilde{\varphi}} + \underline{\tilde{\mathsf{q}}}. \tag{4.8}$$

Using the notation introduced in Section 2.1.2

$$\underline{\tilde{\varphi}} = \left(\mathsf{Y} + \mathsf{B}\Lambda^{-1}\Gamma\right)^{-1}\,\left[\mathsf{B}\Lambda^{-1}\,\left(\underline{\tilde{\mathsf{f}}} + \underline{\tilde{\mathsf{a}}}\right) + \underline{\tilde{\mathsf{q}}}\right] \tag{4.9}$$

If the integral $\mathcal{I}_{\mathcal{V}}$ can be calculated in some way (analytically or numerically), then the vector $\underline{\tilde{\mathsf{q}}}$ is known. The present version of `AcouSTO` can use the formulation A only if monopole sources are present. Other field sources will be included in the forthcoming versions of the code.

Formulation B

In most of the realistic applications, the integral $\mathcal{I}_{\mathcal{V}}$ cannot be easily calculated, and formulation A is not a feasible approach to the problem. In such a case, it is convenient to decompose the acoustic potential in the contributions of the *incident* (or *primary*) and *scattered* fields, $\tilde{\varphi} = \tilde{\varphi}_{\text{in}} + \tilde{\varphi}_{\text{sc}}$. The problem can be reduced to an integral formulation for the scattered field of the form

$$E(\mathbf{y})\,\tilde{\varphi}_{\text{sc}}(\mathbf{y}) = \oint_S \left(G_0\,\tilde{\chi}_{\text{sc}} - \tilde{\varphi}_{\text{sc}}\,\frac{\partial G_0}{\partial n} + s\,\tilde{\varphi}_{\text{sc}} G_0\,\frac{\partial \theta}{\partial n}\right) e^{-s\,\theta}\,d\mathcal{S}(\mathbf{x}) \tag{4.10}$$

The effect of the field sources are included into $\tilde{\chi}_{\text{sc}}$ by decomposing Eq. 4.4 into its incident and scattering components

$$\underline{\tilde{\chi}}_{\text{sc}} = \Lambda^{-1}\,\left(\underline{\tilde{\mathsf{f}}} + \underline{\tilde{\mathsf{a}}}\right) - \Lambda^{-1}\Gamma\,\underline{\tilde{\varphi}}_{\text{in}} - \Lambda^{-1}\Gamma\,\underline{\tilde{\varphi}}_{\text{sc}} - \underline{\tilde{\chi}}_{\text{in}} \tag{4.11}$$

Note that the incident field is the acoustic field induced by the field sources: it represents the acoustic field that would be present if all the scattering obstacles were absent. Thus, it should be known in advance in order to make the problem solvable. The final form of the solving system is

$$\underline{\tilde{\varphi}}_{\text{sc}} = \left(\mathsf{Y} + \mathsf{B}\Lambda^{-1}\Gamma\right)^{-1}\,\left[\mathsf{B}\Lambda^{-1}\,\left(\underline{\tilde{\mathsf{f}}} + \underline{\tilde{\mathsf{a}}}\right) - \mathsf{B}\Lambda^{-1}\Gamma\,\underline{\tilde{\varphi}}_{\text{in}} - \mathsf{B}\underline{\tilde{\chi}}_{\text{in}}\right]. \tag{4.12}$$

Example: Scattering of the field generated by a point source by a sound hard surface

In order to better clarify the equivalence of the two approaches, and to make some consideration about their numerical advantages and drawbacks, let's consider the simple example of a single point source in $\mathbf{x}_s$, radiating into a field where an acoustically impermeable surface is present. The boundary condition associated to such a surface is

$$\frac{\partial \tilde{\varphi}}{\partial n} = 0 \tag{4.13}$$

In this case, $\gamma = 0$, $\lambda = 1$, $\tilde{f} = 0$, and $\mathbf{g} = \mathbf{0}\ \forall \mathbf{x} \in \mathcal{S}$, and thus $\mathsf{\Gamma} = \mathsf{0}$ and $\mathsf{\Lambda} = \mathsf{I}$. Furthermore, being $q(\mathbf{x}, \kappa) = \hat{q}(\kappa)\, \delta(\mathbf{x} - \mathbf{x}_s)$, the field integral can be solved analytically

$$\mathcal{I}_{\mathcal{V}}(\mathbf{y}, \kappa) = \hat{q}(\kappa)\, G_0(\mathbf{y}, \mathbf{x}_s)\, e^{-s\,\theta_s}, \tag{4.14}$$

with $\theta_s = |\mathbf{x}_s - \mathbf{y}|/c_0$. As a consequence, the entries of the vectors $\underline{\tilde{\mathsf{q}}}$, $\underline{\tilde{\varphi}}_{\mathsf{in}}$ and $\underline{\tilde{\chi}}_{\mathsf{in}}$ can be evaluated analytically as

$$\tilde{\mathsf{q}}_j = \hat{q}(\kappa)\, G_0(\mathbf{x}_j, \mathbf{x}_s)\, e^{-s\,\theta_{sj}}, \quad \theta_{sj} = \frac{|\mathbf{x}_s - \mathbf{x}_j|}{c_0}, \qquad \text{for } j = 1, ..., N. \tag{4.15}$$

$$\underline{\tilde{\varphi}}_{\mathsf{in}} = \underline{\tilde{\mathsf{q}}} \tag{4.16}$$

$$[\underline{\tilde{\chi}}_{\mathsf{in}}]_j = \hat{q}(\kappa)\, \nabla \left[G_0(\mathbf{x}_j, \mathbf{x}_s)\, e^{-s\,\theta_{sj}} \right] \cdot \mathbf{n}_j \tag{4.17}$$

The form of the final system of equations for the two approaches is

- **Formulation A:**

$$\underline{\tilde{\varphi}} = \mathsf{Y}^{-1}\, \underline{\tilde{\mathsf{q}}} \tag{4.18}$$

- **Formulation B:**

$$\underline{\tilde{\varphi}} = \underline{\tilde{\varphi}}_{\mathsf{sc}} + \underline{\tilde{\varphi}}_{\mathsf{in}} = -\mathsf{Y}^{-1}\, \mathsf{B}\, \underline{\tilde{\chi}}_{\mathsf{in}} + \underline{\tilde{\varphi}}_{\mathsf{in}}. \tag{4.19}$$

Formulation B requires the evaluation of the incident field and its normal derivative at the collocation points, whereas formulation A requires only the evaluation of $\underline{\tilde{\mathsf{q}}}$, which acts directly as the known term of the system. Considering that $\underline{\tilde{\mathsf{q}}} \equiv \underline{\tilde{\varphi}}_{\mathsf{in}}$, the formulation A is clearly the most convenient in this case. Indeed, for the same discretization of the boundary, formulation A yields to a system having a known term, $\underline{\tilde{\mathsf{q}}}$, analytically evaluated. On the contrary, in the formulation B the known term of the system is $\mathsf{B}\, \underline{\tilde{\chi}}_{\mathsf{in}}$, which is a numerical approximation of the source integral associated to the normal derivative of the scattered field on $\mathcal{S}$, whose accuracy depends on the mesh.

4.2.2 *Dirichlet–type* boundary conditions

When is $\tilde{\varphi}$ the known part of the Cauchy data of the BVP, Eq. 4.2 must be rewritten in the form

$$\underline{\tilde{\varphi}} = \Gamma^{-1} \left(\underline{\tilde{f}} + \underline{\tilde{a}} \right) - \Gamma^{-1} \Lambda \, \underline{\tilde{\chi}} \tag{4.20}$$

The resulting solving system of equations for $\underline{\tilde{\varphi}}$ for the two formulation is derived in the following.

Formulation A

Equation 4.20 can be combined with 4.7 to obtain a system of equations for $\underline{\tilde{\chi}}$, which now is the unknown part of the Cauchy set. Using the same symbols introduced in Section 2.1.2, it can be easily shown that the result is

$$\underline{\tilde{\chi}} = \left(B + Y\Gamma^{-1}\Lambda \right)^{-1} \left[Y\Gamma^{-1} \left(\underline{\tilde{f}} + \underline{\tilde{a}} \right) - \underline{\tilde{q}} \right] \tag{4.21}$$

Again, if the integral $\mathcal{I}_{\mathcal{V}}$ can be calculated in some way (analytically or numerically), then the vector $\underline{\tilde{q}}$ is known.

Formulation B

Using the same approach as in Section 4.2.1, the effect of the field sources are included into $\tilde{\varphi}_{sc}$ by decomposing Eq. 4.20 into its incident and scattering components

$$\underline{\tilde{\varphi}}_{sc} = \Gamma^{-1} \left(\underline{\tilde{f}} + \underline{\tilde{a}} \right) - \Gamma^{-1} \Lambda \, \underline{\tilde{\chi}}_{sc} - \Gamma^{-1} \Lambda \, \underline{\tilde{\chi}}_{in} - \underline{\tilde{\varphi}}_{in} \tag{4.22}$$

Also in this case, the incident field must be known to solve the problem as

$$\underline{\tilde{\chi}}_{sc} = \left(B + Y\Gamma^{-1}\Lambda \right)^{-1} \left[Y\Gamma^{-1} \left(\underline{\tilde{f}} + \underline{\tilde{a}} \right) - Y\Gamma^{-1}\Lambda \, \underline{\tilde{\chi}}_{in} - Y \, \underline{\tilde{\varphi}}_{in} \right]. \tag{4.23}$$

Example: Scattering of the field generated by a point source by a sound–soft surface

Now, let's consider the simple example of a single point source in $\mathbf{x}_s$, radiating into a field where a sound–soft surface is present. The boundary condition associated to such a surface is

$$\tilde{\varphi} = 0 \tag{4.24}$$

In this case, $\gamma = 1$, $\lambda = 0$, $\tilde{f} = 0$, and $\mathbf{g} = \mathbf{0}$ $\forall \mathbf{x} \in \mathcal{S}$, and thus $\Gamma = I$ and $\Lambda = 0$.As in the example 4.2.1, the field integral gives

$$\mathcal{I}_{\mathcal{V}}(\mathbf{y}, \kappa) = \hat{q}(\kappa) \, G_0(\mathbf{y}, \mathbf{x}_s) \, e^{-s\,\theta_s}, \tag{4.25}$$

with $\theta_s = |\mathbf{x}_s - \mathbf{y}|/c_0$. As a consequence, the entries of the vector $\underline{\tilde{\mathsf{q}}}$ can be evaluated as

$$\tilde{\mathsf{q}}_j = \hat{q}(\kappa)\, G_0(\mathbf{x}_j, \mathbf{x}_s)\, e^{-s\,\theta_{sj}}, \quad \theta_{sj} = \frac{|\mathbf{x}_s - \mathbf{x}_j|}{c_0}, \qquad \text{for } j = 1, ..., N. \tag{4.26}$$

The form of the final system of equations for the two approaches is

- **Formulation A:**

$$\underline{\tilde{\chi}} = \mathsf{B}^{-1}\, \underline{\tilde{\mathsf{q}}} \tag{4.27}$$

- **Formulation B:**

$$\underline{\tilde{\chi}} = \underline{\tilde{\chi}}_{\mathsf{sc}} + \underline{\tilde{\chi}}_{\mathsf{in}} = -\mathsf{B}^{-1}\mathsf{Y}\, \underline{\tilde{\varphi}}_{\mathsf{in}} + \underline{\tilde{\chi}}_{\mathsf{in}}\,. \tag{4.28}$$

Also in this case, and for the same reasons explained at the end of Section 4.2.1, formulation A is clearly convenient.

4.2.3 Wall impedance

`AcouSTO` can take into account impedent walls by providing, with the format specified in Section 4.1, the local value of the *complex impedance*,

$$\mathcal{Z}(\mathbf{x}, s) = \mathcal{R}(\mathbf{x}, s) + j\,\mathcal{X}(\mathbf{x}, s), \qquad \mathbf{x} \in \partial\mathcal{V}, \tag{4.29}$$

being $\mathcal{R}$ and $\mathcal{X}$ are the resistive and reactive parts, respectively. The wall acoustic mobility (admittance) is $\mathcal{A} = 1/\mathcal{Z}$. It is possible to associate a single complex value of $\mathcal{Z}$ to the whole boundary, or to specify the value of $\mathcal{Z}$ for a subset of panels (see Section 4.1). The influence of $\mathcal{Z}$ on the solution is consistent with the assumption of a "locally reacting" surface.

It is worth noting that the acoustic properties of a material are often defined in the literature in terms of its "reflection coefficient" R, which is a complex quantity related to $\mathcal{Z}$ through the formula (see, *e.g.*, [1])

$$\mathcal{Z}(\mathbf{x}, s)\, \cos(\theta_I) = \rho\, c\, \frac{1 + R(\mathbf{x}, s)}{1 - R(\mathbf{x}, s)} \tag{4.30}$$

where the angle θ_I is the incidence angle of the incoming (planar) wave front, and is equal to zero when the wave front impinges normally to the boundary, *i.e.*, when for the wave unit vector holds $\hat{\mathbf{w}} = -\hat{\mathbf{n}}$. Note also that the reflection coefficient is related to the "absorption coefficient" α through

$$\alpha = 1 - |R(\mathbf{x}, s)|^2 \tag{4.31}$$

which is a real number widely used in engineering applications.

Starting from the physical definition of $\mathcal{Z}$

$$\mathcal{Z} = \frac{\tilde{p}}{\mathbf{v} \cdot \hat{\mathbf{n}}_{in}} \tag{4.32}$$

(where $\hat{\mathbf{n}}_{in}$ is the unit normal pointing into the wall) and making use of the linearized Bernoulli's theorem, $\tilde{p} = -\rho\, s\, \tilde{\varphi}$, Eq. 4.1 can be specialized for the case of a locally impedant wall. Considering that

$$\mathbf{v} \cdot \hat{\mathbf{n}}_{in} = -\mathbf{v} \cdot \hat{\mathbf{n}} = -\frac{\partial \tilde{\varphi}}{\partial n} \tag{4.33}$$

one obtains

$$\frac{\partial \tilde{\varphi}}{\partial n} - \frac{\rho\, s}{\mathcal{Z}}\, \tilde{\varphi} = 0, \tag{4.34}$$

which corresponds to $\gamma = -\rho\, s/\mathcal{Z}$, $\lambda = 1$, $f = 0$, and $\mathbf{g} = \mathbf{0}$ in Eq. 4.1. As a consequence, $\Lambda = \mathsf{I}$ and

$$\Gamma = -\rho\, s \begin{bmatrix} 1/\mathcal{Z}(\mathbf{x}_1) & 0 & 0 & 0 \\ 0 & 1/\mathcal{Z}(\mathbf{x}_2) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 1/\mathcal{Z}(\mathbf{x}_N) \end{bmatrix} \tag{4.35}$$

$$= -\rho\, s \begin{bmatrix} \mathcal{A}(\mathbf{x}_1) & 0 & 0 & 0 \\ 0 & \mathcal{A}(\mathbf{x}_2) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \mathcal{A}(\mathbf{x}_N) \end{bmatrix} \tag{4.36}$$

4.3 Setup of boundary conditions

`AcouSTO` gives to the user full control of the boundary conditions to implement the general formulation presented so far. The user can provide arbitrary values to the functions $\gamma(\mathbf{x}, \kappa)$, $\lambda(\mathbf{x}, \kappa)$, and $\tilde{f}(\mathbf{x}, \kappa)$, and to the vector $\mathbf{g}(\kappa)$ using three different approaches for both Neumann– or Dirichlet–type formulations.

The input variables controlling this feature are `dirneu` and `custombc`, in the `modsol` configuration block. In addition, `knw` controls the formulation used (A or B) for the

primary field. The values these variables can assume are:

$$\texttt{dirneu} = \begin{cases} 1 & \text{Dirichlet–type boundary conditions.} \\ 2 & \text{Neumann–type boundary conditions. This is the default.} \end{cases}$$

$$\texttt{knw} = \begin{cases} 1 & \text{Formualtion A for source terms. This is the default.} \\ 2 & \text{Formualtion B for source terms.} \end{cases}$$

$$\texttt{custombc} = \begin{cases} 0 & \text{Legacy format (see Section 4.3.1). This is the default.} \\ 1 & \text{Custom b.c. using AcouSTO files (Section 4.3.2).} \\ 2 & \text{Custom b.c. using Gmsh file (Section 4.3.3).} \end{cases}$$

The three options for `custombc` refer to the three different approaches to provide the values of boundary properties, which are described in the following sections.

The use of `custombc` = 1/2 is strongly encouraged. The possibility to use the legacy boundary conditions input scheme will be removed in one of the future releases.

4.3.1 `custombc` =0. Legacy format for impedent and radiant panels

The input structure presented in Section 4.3 can be used to describe any type of boundary conditions. The only drawback is that the acousto–mechanical properties of the boundary could appear somehow hidden by the general formulation adopted (see Section 4.2.3).

Although the use of the scheme presented in Section 4.3 is strongly encouraged, starting from v1.6, AcouSTO retains the possibility to explicitly define the acoustic impedance of the boundary (or of part of it) and/or the presence of radiant panels by listing them in files with specified names with a format defined in Section B.2.2. The input variables dedicated to this are presented in Tables 4.3 and 4.4. If the variable `nimped` and `nradian` are negative, then the surface is assumed to be uniformly impedent and/or radiant. In this case the values of the impedance and the radiation amplitude are rerad from the config file (see Section A.2).

4.3.2 `custombc` =1. The AcouSTO boundary condition files

When this option is activated, it is also necessary to provide the names of the files collecting the values of the boundary conditions functions by including the following entries in the `modsol` block

Table 4.3: **Impedent panels**

variable	config block	meaning
`nimped`	`modsol`	number of impedent panels (`nimped`<0 implies a uniform radiation for the whole boundary)
`imped_file`	`modsol`	name of the file containing the indices and the complex impedance of the impedent panels

Table 4.4: **Radiant panels**

variable	config block	meaning
`nradian`	`modsol`	number of radiant panels (`nradian`<0 implies a uniform radiation for the whole boundary)
`radiants_file`	`modsol`	name of the file containing the indices of the radiant panels and the complex amplitude of their motion

```
bc_lambda_file = "name_of_lambda_file"

bc_gamma_file = "name_of_gamma_file"

bc_func_file = "name_of_func_file"

bc_g_file = "name_of_gvec_file"
```

Note that the current version of the code can handle only a $\mathbf{g}$ vector constant in space. Thus, the `bc_g_file` must contain only one line with the format described in Section B.2 (if the file is longer, then only the first line is read).

Note that the diagonal matrices Γ and Λ used in the previous Sections were introduced for notational simplicity and are not actually built by the code. Thus, no memory is waisted to store all those zeroes.

Since all the functions in Eq. 4.1 depend on frequency, you may want to specify different input files in case of a multi-frequency analysis. To do this, you have to comply to a simple naming convention. The name of each file must include an integer number representing the frequency index the file refers to, and the entries of the configuration file must have a `%d` where the frequency index is expected. For example, let's assume that we want analyze three frequency, and that the files containing the function λ are `lambda_1, lambda_2,`

lambda_3. The correct configuration entry is

```
bc_lambda_file = "lambda_%d"
```

The file format is specified in Section B.2. Note that, although constant in space, **g** can vary with frequency.

Note that AcouSTO doesn't check for the values of γ and λ to be compatible with the user selection of Neumann or Dirichlet conditions.
The coherence of the input is responsibility of the user. Specifically, the following conditions must be satisfied to avoid inconsistent results

$$\forall\, \mathbf{x} \in \partial\mathcal{V} \begin{cases} \lambda(\mathbf{x}, \kappa) \neq 0 & \text{Neumann–type} \\ \gamma(\mathbf{x}, \kappa) \neq 0 & \text{Dirichlet–type} \end{cases} \tag{4.37}$$

4.3.3 custombc =2. Boundary conditions from Gmsh mesh files

AcouSTO 2.0 introduces the possibility to define custom boundary conditions using Gmsh . This feature enables an easy and powerful way to prepare the input files in presence of complex boundaries which exhibit non uniform acoustic properties. Using this input scheme for the acoustic properties of the boundary, the bc_* files are generated by AcouSTO on the basis of the entries found in the configuration file. Indeed, the input schemes is based on the definition of *physical groups* in Gmsh . Each *physical groups* must be associated to a particular acoustic response of the boundary. The acoustic properties of each physical group have to be defined in the configuration file by setting a specific value for γ, λ, and f to the tag associated to the physical groups (see Gmsh manual for tags definition). To this aim, the physical groups are listed within the gmshbc list as

```
gmshbc = ["a_phys_group", "another_phys_group", ...];
```

For each name in the list, the corresponding input block must be present, with the syntax

```
  a_phys_group={
        tag = <tag of physical group>;
        re_lambda = real part of λ;
        im_lambda = imaginary part of λ;
        re_gamma  = real part of γ;
        im_gamma  = imaginary part of γ;
        re_f      = real part of f;
        im_f      = imaginary part of f;
};
```

Once that the properties associated to each physical group are read, `AcouSTO` produces the boundary conditions files according to the fixed naming scheme

```
bc_lambda_file = "exported_gmshbc_lambda"

bc_gamma_file = "exported_gmshbc_gamma"

bc_func_file = "exported_gmshbc_func"
```

In case of boundary properties depending on frequency, the names of physical group must include `%d` at the end. The number of boundary properties for each physical group must be equal to the number of frequencies for all the physical groups. For example, the correct configuration file section for two groups and two frequencies must be as follows:

```
gmshbc = ["a_phys_group%d", "another_phys_group%d"];

a_phys_group0={
tag = <tag of physical group>;
re_lambda = real part of λ;
im_lambda = imaginary part of λ;
re_gamma  = real part of γ;
im_gamma  = imaginary part of γ;
re_f      = real part of f;
im_f      = imaginary part of f;
};
a_phys_group1={
tag = <tag of physical group>;
re_lambda = real part of λ;
im_lambda = imaginary part of λ;
re_gamma  = real part of γ;
im_gamma  = imaginary part of γ;
re_f      = real part of f;
im_f      = imaginary part of f;
};
another_phys_group0={
tag = <tag of physical group>;
re_lambda = real part of λ;
im_lambda = imaginary part of λ;
re_gamma  = real part of γ;
im_gamma  = imaginary part of γ;
re_f      = real part of f;
im_f      = imaginary part of f;
};
another_phys_group1={
tag = <tag of physical group>;
re_lambda = real part of λ;
im_lambda = imaginary part of λ;
re_gamma  = real part of γ;
im_gamma  = imaginary part of γ;
re_f      = real part of f;
```

```
im_f       = imaginary part of f;
};
```

In such a case, the exported files will be named as

```
exported_gmshbc_lambda1
exported_gmshbc_lambda2
exported_gmshbc_gamma1
exported_gmshbc_gamma2
exported_gmshbc_f1
exported_gmshbc_f2
```

The input of vector **g** remans the same as for `custombc = 1` (see Section 4.3.2). The use of this input method makes the input files much more compact and readable, being now the attention focused on boundary's subdomains, rather than on single elements. Details on this input method are given in Section B.2.3 and in the dedicated tutorial.

4.4 Memory management

4.4.1 The KROW flag

When dealing with very large problems, a large amount of memory is required to complete the analysis. Usually, when a single process takes control of a large part of the available RAM, the performances of the machine dramatically drops down. As a consequence, the calculation time increases, as well as the risk of critical errors and machine hang-ups. To avoid this situation (or, at least, to move it further away) the code includes a simple mechanism of memory management.

The integral coefficients, once evaluated by the dedicated module, are stored on unformatted files. When the BEM matrices need to be assembled, the coefficients are retrieved from the disk files, and stored in RAM in form of three-dimensional arrays. The input variable *KROW* controls the dimensions of these arrays, by controlling the number of matrix rows to be loaded at a time. The way it works is described in Table 4.5.

Value	No. of matrix rows loaded in RAM
$1 \leq$ KROW $<$ NCNTR	KROW
KROW ≤ 0	NCNTR (full problem)
KROW $\geq$ NCNTR	NCNTR (full problem)

Table 4.5: Use of the variable KROW to control memory allocation.

From Table 4.5, we see that the user can choose the amount of matrix rows to load in RAM from 1 up to the entire problem. The optimal choice strongly depends on the characteristics of the system that you are using. When setting the $KROW$ value, keep in mind that a reduction in memory allocation results in a higher number of access to disk. The more appropriate trade-off is a matter of trial and errors.

Versions of `AcouSTO` $\geq$ 1.2 accept the runtime option `-m` to display an estimate of the memory required by the run.

4.4.2 Stored or recalculated coefficients

Version 1.3-beta includes a new feature to save memory space. Up to version 1.2-beta, the integral coefficients representing the influence of each panel onto collocation points and microphones were evaluated in the dedicated module, and kept in memory as very large arrays. For intensive simulations, these arrays can be really huge and rapidly saturate the RAM even for the most powerful clusters.

From version 1.3-beta, by default the integral coefficients are evaluated at the moment of their use and discarded. This approach results in a significant reduction of the memory allocated, with a slight reduction of peak performances in all those cases where the same coefficient is used more than once to complete the calculation (not very frequent, indeed).

In any case, the possibility to store the coefficients into arrays is still available, being this possibility useful during development and debugging. Setting

```
pre_calculated_coefs=1
```

in the configuration file activates this possibility (see Chapter A for details).

Note that the variable `KROW` has (obviously) no effect when the coefficients are not stored in RAM.

4.5 MPI and ScaLAPACK

`AcouSTO` uses the `MPI-2` communication protocol to run in parallel on clusters of computers. In this section, the basic criteria underlying the parallel implementation scheme are briefly described. It should be emphasized that the method we adopted is very simple and preliminary. Thus, the user should not expect top performances and high scalability. The parallel implementation will be improved in the future versions of the code (of course, external contributions are very welcome).

From the point of view of the distribution of computational load, the algorithm can be divided into three parts: *i*) the evaluation of the integral coefficients, Eq. 2.7; *ii*) the solution of the linear system, Eq. 2.8; *iii*) the assembly of the solution in the field, Eq.2.11. The code includes direct calls to the `MPI` functions to distribute the evaluation of the integral coefficients among the nodes. This part of the algorithm is *embarrassingly parallel*, *i.e.*, can be divided into subtasks that do not need to communicate with each other. In the present version of the code, the subtasks are generated by equally partitioning the total work into a number of subtasks equal to the number of MPI processes (SPMD pattern). A more efficient, especially on heterogenous clusters, Master/Worker version is under development. In figure 4.2 we have reported an activity diagram to show the major blocks of the algorithm. In the configuration file (described in Appendix A) there are 4 parameters related to the parallelization of the algorithm:

- `nprows`= P_r number of process rows in the process grid
- `npcols`= P_c number of process columns in the process grid
- `row_blocking_factor` = MB row block size
- `col_blocking_factor` = NB column block size

The ScaLAPACK library expects the processes to be organized in a process grid. In order to make dense matrix computations as efficient as possible, ScaLAPACK uses a data layout called 2–dimensional block cyclic distribution[1]. In short the P processes of our cluster are arranged in a $P_r\, X\, P_c$ rectangular array of processes. The rows and the columns of the process are divided in groups of size MB and NB respectively and distributed in a cyclic manner (fig.4.1). The data contained at the index (j, k) of the matrix is thus stored in process $((j-1)/MB \bmod P_r, (k-1)/NB \bmod P_c)$. The right choice of the rows and column block size is hence important to ensure a well balanced distribution of the data over the

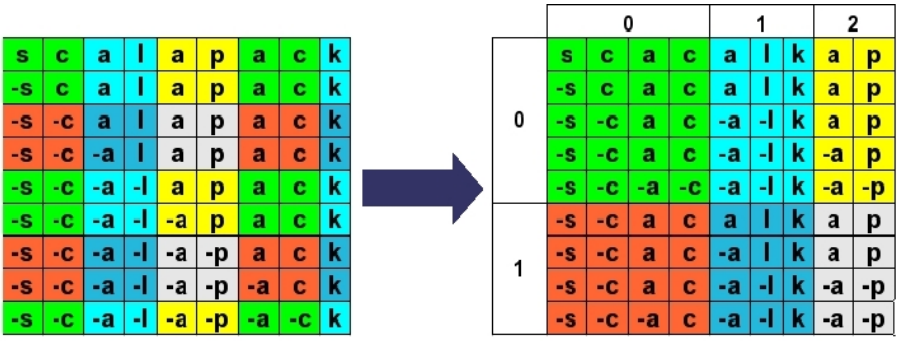


Figure 4.1: ScaLAPACK process grid.

processes in the grid. The ScaLAPACK Users' guide suggests that, for the routines called

[1]To better understand the effect of these parameters we address the reader to the ScaLAPACK Users' Guide, available at http://www.netlib.org/scalapack/slug/index.html

by AcouSTO , a good shape for the grid could be, for square matrices, $P_r \ll P_c$ with row and column block size = 64. Anyway, we suggest you check the literature to optimize the parameters of the ScaLAPACK routines involved or, in alternative, try changing the parameters to achieve the best performance.

4.6 Choose the solver

Version 1.3-beta introduces the possibility to choose the solver of the linear system. Up to version 1.2-beta, the solution is obtained through a pseudoinverse-based solver, available in the ScaLAPACK libraries. This option is the default choice also in version 1.3-beta, but now an iterative *GMRES* solver can be used on demand.

The reason of such an option stems from the higher parallel performance of this kind of solvers for dense matrices (as the ones we deal with). In addition, the number of operation required is proportional to $N^2\, N_{it}$, where N_{it} is the number of iteration to achieve the convergence.

The solver choice is controlled by the input variable `solver`. Specifically

`solver="GMRES";` activates the iterative solver,

whereas

`solver="PSEUDOINV";` switches to the ScaLAPACK solver.

Please note that the GMRES solver has been written from scratch, and is still under validation. A detailed description of the algorithm used will be included soon in this manual.

4.7 Run the code

Once that the the input data structure has been completed according to the instructions given in the previous sections, the code can be run using the `mpirun` command. Here we assume that the `MPI` environment is properly set up, and that the user has complete access to all the nodes involved in the calculation.

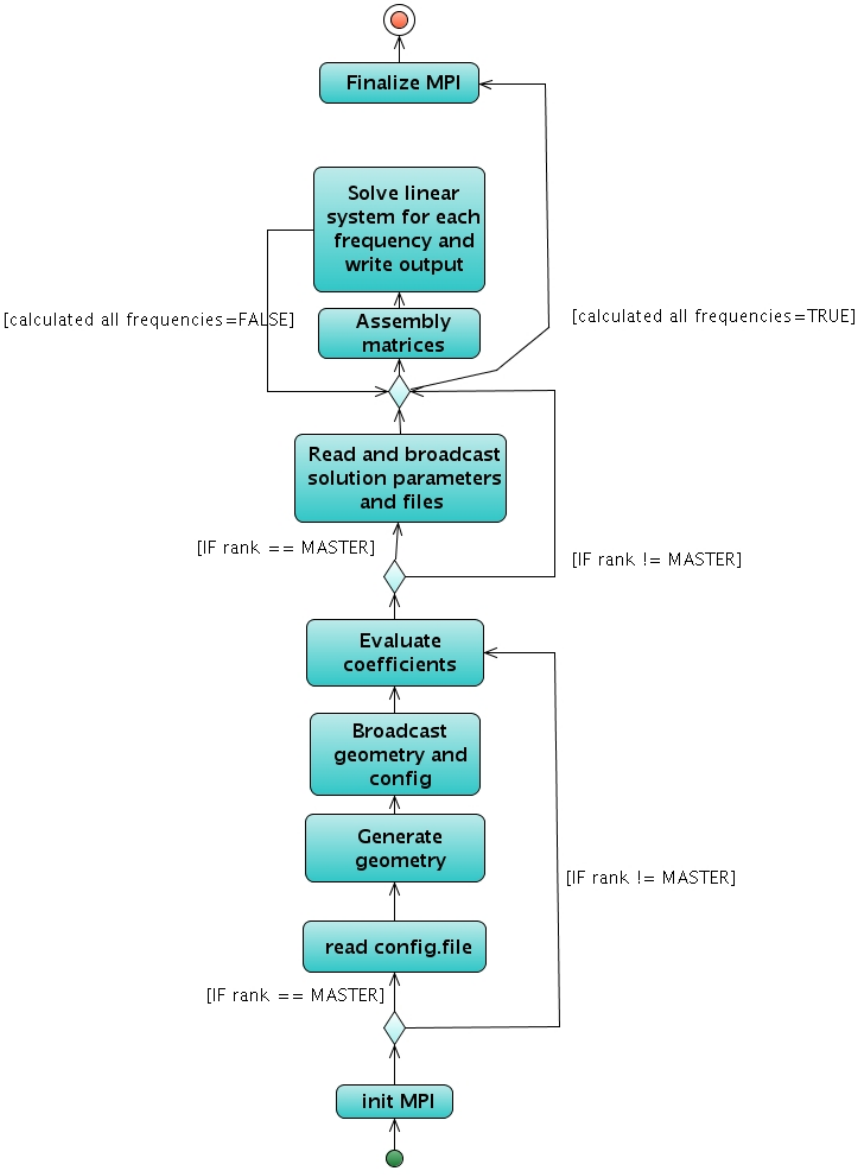


Figure 4.2: Activity diagram of `AcouSTO` .

4.7.1 Running on a single node

In order to run the code on a single computer, even if it doesn't make much sense other than for testing purposes, you can issue the command:

```
mpirun -np <n.proc> acousto -f <config.file>
```

where `<n.proc>` is the number of processes, which depends on the grid size you declared in the configuration file, and `<config.file>` is the configuration file described in Appendix A which must be on the master node.

4.7.2 Running in a distributed environment

In this section we assume you have a cluster of `n` nodes in which `AcouSTO` has been installed and can be found in the run path of each node and that the proper remote execution permissions have been correctly set up. There are many available options for the `mpirun` command line and there are many ways to run the code on a distributed environment; we will give here some basic instructions but please refer to the `mpirun` documentation for further details. In order to run `AcouSTO` on your cluster you have multiple choices, the two most common ways of running an MPI code are described in the following sections.

4.7.3 Running with one process on each node

This is the easiest way to launch the code. Just give the command

```
mpirun -np <n.nodes>\
       -host <node 1>,<node 2>,...,<node n>\
       acousto\
       -f <config.file>
```

and one `AcouSTO` process will start on each node defined with the option `-host`.

4.7.4 Running with different number of processes on each node

You can also choose to run `AcouSTO` starting more than one process on each node. The cleanest way to do this is with the help of a `hostfile` in which you can declare the nodes and the "slots" for each node. If you want, for instance, to run `AcouSTO` with 2 processes on

node_1.acousto.org, 1 process on node_2.acousto.org and 4 processes on node_n.acousto.org you can write a `hostfile` with the following content:

```
node_1.acousto.org slots=2
node_2.acousto.org slots=1
...
node_n.acousto.org slots=4
```

Now we can pass the above file to the `mpirun` executable:

```
mpirun -np <n.proc> --hostfile <hostfile.name> acousto -f <config.file>
```

The number of processes specified with the option `-np` can be omitted and, in this case, `mpirun` will launch "slots" processes on each node as defined in the hostfile[2]

4.8 Data post-processing

Once that the solution is completed without errors, the results can be visualized in several ways, depending on the specific application at hand, and on the user's needs and preferences. A huge amount of reliable public domain software is available for scientific visualization. In order to make possible an easy visualization of the solution also to the users less familiar with this kind of stuff, `AcouSTO` provides the solution in different formats (details in Appendix B), suitable for an easy post–processing using the most widely used codes for scientific visualization. Specifically, the solution on the body surface and at the microphones is written into

- plain text files containing space-separated columns (extension `*.out`);
- ASCII files in "legacy" VTK format (extension `*.vtk`, see http://www.vtk.org) for the field on the boundary and for the pressure at the microphones;
- ASCII files in `Gmsh` v2.2 format (extension `*.msh`, see http://geuz.org/gmsh/) for the field on the boundary and for the pressure at the microphones (only from `AcouSTO v1.6` on) ;
- plain text files in VRML2 format to visualize the geometry of the domain.

[2]You can also limit the number of maximum processes on each node with the option `max_slots` of the hostfile. Please refer to the `man` page of `mpirun` for further details.

All these files follow a naming convention useful to easily retrieve your results.[3] The naming scheme and the files content is given in Table 4.6. Details about the data format is given in Appendix B. In Table 4.6, `<RunName>` is the name of the current run, as

Table 4.6: Naming convention for the output files.

Name	Content
`<RunName>-surface.wrl`	domain boundary in `VRML2` format
`<RunName>-controls.wrl`	control points location in `VRML2` format
`<RunName>-mics.wrl`	microphones location in `VRML2` format
`<RunName>-chief.wrl`	chief points location in `VRML2` format
`<RunName>-sources.wrl`	point sources location in `VRML2` format
`<RunName>-radiants.wrl`	radiant panels location in `VRML2` format
`<RunName>-imped.wrl`	impedent panels location in `VRML2` format
`<RunName>-unormals.vtk`	unit normal vectors on the boundary in `VTK` format
`<RunName>-body.raw`	surface panels in raw–data format to be imported in Blender
`<RunName>-surf-<XX.YYYY>Hz.vtk`	solution on the boundary in `VTK` format
`<RunName>-mics-<XX.YYYY>Hz.vtk`	solution at the microphones in `VTK` format
`<RunName>-surf-<XX.YYYY>Hz.msh`	solution on the boundary in `Gmsh v2.2` format
`<RunName>-mics-<XX.YYYY>Hz.msh`	solution at the microphones in `Gmsh v2.2` format
`<RunName>-surf-<XX.YYYY>Hz.out`	solution on the boundary in space—separated text format
`<RunName>-mics-<XX.YYYY>Hz.out`	solution at the microphones in space—separated text format
`<RunName>-fresp-cntZZ.out`	frequency response at control point `ZZ` in space—separated text format
`<RunName>-fresp-micZZ.out`	frequency response at microphone `ZZ` in space—separated text format

provided in the `runinfo` block of the configuration file (see Appendix A), `<XX.YYYY>` is the value of the frequency in Hz, `<ZZ>` represents the indices of the control points and microphones for which the frequency response is required. Note that there no limitation in the number of frequencies that can be solved in a single run.

[3] To help the user in retrieving data from a large archives, such those produced by heavy simulation campaigns, extensive parametric analyses, or simply by a long-lasting use of the code, an interface to the `MySQL` database is under development. An early experimental version is included in this distribution.

Starting from version AcouSTO v1.6, the output files are saved under a single directory. The default location of the directory is the where the config file is, and its name follows the convention <RunName>-YYYY-MM- DD. Location and name of the output directory can be modified using the dedicated input variables (see Appendix A). In particular, it is possible to change the time format using the variable dirsuffix and standard time format strings. For example, to append the time of the run in the format HH-MM-SS, include in the runinfo block the declaration `dirsuffix="%F-%H-%M-%S";`

For complex geometries, when a large number of frequencies is analyzed, the output of AcouSTO produces a large number of files, which can take a significant amount of disk space, and make your working directory a mess quite soon (a well organized mess, but always a mess). To keep your working space clean, the output on the VRML, the frequency response, and the VTK and Gmsh files can be deactivated through the configuration flags printwrl, printfresp, printout, printmsh, and printvtk, respectively.[4]

The *.out files can be used directly to generate 2D and 3D plots using simple plotting programs. We use primarily gnuplot (http://www.gnuplot.info) or Grace (http://plasma-gate.weizmann.ac.il/Grace/), both available for free on all the hardware/software platforms. In addition, such a simple format it can be easily read by ancillary codes produced by the user and further processed to meet specific requirements, without modification of the AcouSTO sources. An example of the pictures that can be obtained using this format is given in Fig. 4.3.

[4] **Warning to users:** not all the output-control flags are active in the beta version of the code.

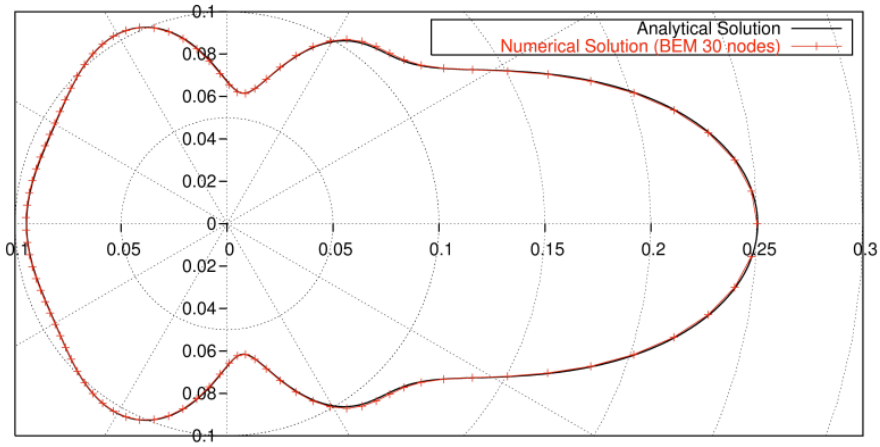


Figure 4.3: Polar plot obtained using `gnuplot` on the `AcouSTO` output without additional post–processing. Scattering of a plane wave by a sphere of unit radius ($ka = \pi$ plus CHIEF regularization).

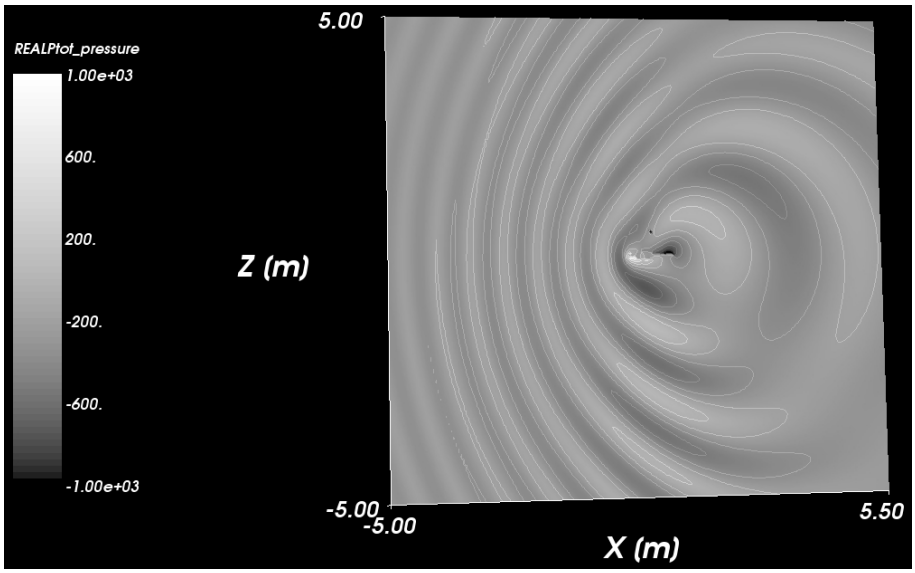


Figure 4.4: 3D visualization obtained using `mayavi` and the *.vtk `AcouSTO` output files, without additional post–processing.

`AcouSTO` has no limitations in the complexity of the geometry of the domain. In order to visualize effectively the three–dimensional acoustic field, we included the possibility to

save the solution in VTK format (http://www.vtk.org), and the geometry of the boundary in VRML2 format (http://www.web3d.org). The files so obtained can be read by a number of platform–independent, open–source tools, to produce impressive high–quality pictures of your simulations. The authors use primarily mayavi (http://mayavi.sourceforge.net/) and paraview (http://www.paraview.org), but any other software compatible with these formats should work as well.

If the basic visualization features enabled in the AcouSTO release doesn't match the user's need, the source code has to be modified in order to obtain the output in the desired format. The programmer's manual is under development, and will be published soon.

Bibliography

[1] A. D. Pierce, Acoustics - An Introduction. Its Physical Principles and Applications, Acoustical Society of America, New York, 1989.

[2] A. Hirshberg and S.W. Rienstra, An Introduction to Acoustics. Available at http://www.win.tue.nl/ sjoerdr/papers/boek

[3] Harry A. Schenck, Improved Integral Formulation for Acousti Radiation Problems, Journal of the Acoustical Society of America, vol. 44, no. 1, pp. 41–89, 1967.

[4] A. F. Seybert and T. K. Rengarajan, The use of CHIEF to obtain unique solutions for acoustic radiation using boundary integral equations, Journal of the Acoustical Society of America, vol. 81, no. 5, pp. 1299-1306,1987.

[5] M. Gennaretti and U. Iemma, CHIEF regularization approach for aeroacoustoelastic analysis in state-space format, *Journal of Fluids and Structures*, vol. 17, n. 7, pp. 983-999, 2003.

[6] Colton, D. and Kress, R., Integral Equation Methods in Scattering Theory, John Wiley & Sons, New York, 1983.

[7] Morino, L., Is There a Difference Between Aeroacoustics and Aerodynamics? An Aeroelastician's Viewpoint, *AIAA Journal*, vol. 41, part 7, pagg. 1209–1223, 2003

[8] Iemma, U. and Gennaretti, M., A Boundary-Field Integral Equation for identification of acoustic spectrum of cavities, AIAA Paper 2003-3161, 2003.

[9] Morino, L., Boundary Integral Equations in Aerodynamics, *Applied Mechanics Review*, vol. 46, n. 8, pagg. 445-466, 1993

[10] Baker, A.H., Jessup, E. R., Kolev, T. V., A simple strategy for varying the restart parameter in GMRES(m), *Journal of Computational and Applied Mathematics*, 2007.

[11] Burton, A. J. and Miller, G., The application of integral equation methods for numerical solution of exterior boundary value problems, *Proceedings of the Royal Society*, London, vol. A232, page. 201-210, 1971.

APPENDIX A

Configuration file reference

This is where you learn about how to edit the configuration file

A.1 Command line option

The configuration file to be used is set with the command line option

```
-f <cfgfile> or  --dat-file=<cfgfile>
```

The presence of this option is mandatory to run the code. The other valid options for AcouSTO are:

```
-V, --version                print version and exit
-h, --help                   print help
-l, --log-level <n>          set log level(defaults to INFO) : 0 - ERROR
                                                               1 - WARN
                                                               2 - INFO
                                                               3 - DEBUG
```

A.2 Configuration file syntax

The configuration file must include the following blocks:

A.2.1 runinfo

The "runinfo" block contains general parameters and information about the run:

```
runinfo = {
 title  = <title of the run>;
 owner  = <name of the run's owner>;
 kdebug = <debug parameter (unused)>;
 ksymmi = <symmetry flag>;
 vsound = <sound speed, 343 m/s for standard air>;
 krow   = <number of matrix rows to be loaded>;

 nprows = <number of rows of process grid>;
 npcols = <number of cols of process grid>;

 row_block_size= <blocking factor for rows>;
 col_block_size= <blocking factor for cols>;
```

```
 out_filedir= <name of the output files parent directory>;
 out_filepath= <location path of the parent directory>;
 dirsuffix= <time stamp suffix of the parent directory>;

};
```

If you do not specify the number of process rows and columns for the process grid, AcouSTO will try to calculate the grid dimensions using the available number of processes.

A.2.2 modgeom

The "modgeom" block is used to define the geometries of the problem:

```
modgeom = {
 active     = <geometry generation active>;
 outbody    = <output flag (unused)>;
 outcntr    = <output flag (unused)>;
 outnormal  = <output flag (unused)>;
 outmics    = <output flag (unused)>;
 geoms      = [ <list of names of geometry sections separated by a comma> ];

 chief_file =  <name of file with CHIEF points>;
 mics_file  =  <name of file with microphones coordinates>;
 nchief     =  <number of chief points in chief_file>;
 nmics      =  <number of microphones in mics_file>;

};
```

the geometries are defined in different sections the names of which are listed in the "geoms" parameter.

Example:

```
modgeom={
   active = 1;
   geoms  = ["sphere1","nodes1"]; # this defines two blocks named
                                  # sphere1 and nodes1 and defined below
   ...
}
...
sphere1={
```

```
  type        = "sphere";    # a sphere
  radius      = 1.0;         # with radius=1
  segments    = 8;           # 8 segments
  rings       = 9;           # and 9 rings = 72 elements
}
nodes1={
  type        = "nodes";      # a nodes block
  filename    = "mynodes.in"; # defined in mynodes.in file
  nnodb       = 128;          # 128 nodes
  nelmb       = 72;           # 72 elements
  ncntr       = 72;           # 72 control points
  translation = { x=10.0; y=0.0; z=0.0;}; # translation
  rotation    = { x=0.0; y=0.0; z=30.0;}; # no rotation
}
```

Acousto can generate different types of geometries or read a geometry from a file with fixed format. The available types for the geometries are:

- sphere
- cylinder
- plate
- nodes
- gmsh

the syntax for the sections of the different geometries type is described in the following sections.

Sphere

The syntax for the block defining a sphere is:

```
sphere1={  # geometry name ( "geoms" parameter of the modgeom block)
  type        = "sphere";
  radius      = <radius>;
  segments    = <n.of segments>;
  rings       = <n.of rings>;
}
```

Note that with `segments` e indicate the number of slices along the longitude of the sphere, whereas `rings` indicate the number of parallel lines along the latitude. This unusual conventions was chosen to comply with the one used by Blender.

Cylinder

The syntax for the block defining a cylinder is:

```
cyl1={  # geometry name ( "geoms" parameter of the modgeom block)
  type        = "cylinder";
  radius      = <radius>;
  segments    = <n.of segments>;
  rings       = <n.of rings>;
  length      = <length of the cylinder>;
}
```

Plate

The syntax for the block defining a plate is:

```
plate1={  # geometry name ( "geoms" parameter of the modgeom block)
  type        = "plate";
  l1 = <X length>;
  l2 = <Y length>;
  n1 = <n. of elements in X>;
  n2 = <n. of elements in Z>;
}
```

It is important to notice that the `plate` geometry generates an open surface that cannot be used as an isolated body. Each object in a BEM simulations must be a bounded by a closed surface. The `plate` geometry is meant to be a tool to generate a closed surface using multiple plates, properly located and rotated (*e.g.*, six equal plates can form a cube).

Nodes

The syntax for the block defining a generic geometry read from a nodes file in `AcouSTO` native format is:

```
nodes1={  # geometry name ( "geoms" parameter of the modgeom block)
```

```
  type        = "nodes";
  filename    = <name of the file where the nodes are>;
  nnodb       = <number of nodes>;
  nelmb       = <number of elements>;
  ncntr       = <number of control points to be generated>;
}
```

Gmsh

The syntax for the block defining a generic geometry read from a `Gmsh` file is:

```
gmsh1={  # geometry name ( "geoms" parameter of the modgeom block)
  type        = "gmsh";
  filename    = <name of the file where the nodes are>;
  symvec      = {xcomp; ycomp; zcomp;};   #  vector for symmetrization
  refpoint     = {xcoor; ycoor; zcoor;};   #  reference point for symmetrization
  }
```

Scaling and rigid transformations

Note that all the geometries (sphere, nodes, gmsh, cylinder and plate) can be scaled, moved and rotated simply by adding to their input blocks the entries

```
mygeom={
  ...
  scale = {
        x = <x scaling factor>;
        y = <y scaling factor>;
        z = <z scaling factor>;
  };
  translation = {
        x = <x translation>;
        y = <y translation>;
        z = <z translation>;
  };
  rotation = {
        x = <rotation around X axis in degrees>;
        y = <rotation around Y axis in degrees>;
        z = <rotation around Z axis in degrees>;
  };
}
```

The nodes are first scaled, then rotated around x, y, and z in the order, and finally translated.

A.2.3 modsol

The "modsol" block defines the solution parameters:

```
modsol={
 active          =  <solution active>;

 solver="GMRES" or "PSEUDOINV";  <controls the solver type>

 tolerance=1E-6; <tolerance for iteration convergence check>
 maxiterations=1000; <maximum number of GMRES iteration>
 pre_calculate_coefs=1 <activates storage of coefficients in RAM>;

 rho           = <density, 1.225 Kg/m^3 for standard air>;
 nsig          = <number of samples for sigma>;
 minsig        = <min value for sigma>;
 maxsig        = <max value for sigma>;
 nome          = <number of samples for omega>;
 minome        = <min value for omega - in rad/sec >;
 maxome        = <max value for omega - in rad/sec >;
 minfreq        = <min value for frequency - in Hz >;
 maxfreq        = <max value for frequency - in Hz >;

 nsourc          = <number of sources>;     ! Sources and primary field
 sources_file    = <name of file with sources coordinates and frequency >;

 nplanw         = <number of planar waves>;
 planw_file     = <name of planar waves file >;

 dirneu           = (1/2) <Dirichlet (1) or Neumann (2) boundary conditions>;

 knw              = (1/2) <Formulation A (1) or formulation B (2))>;

 custombc    =(0/1/2); <Format of boundary conditions input files>

# <data needed if custombc=0 (or absent)- begin>
  nimped          = <number of impedent panels>;
  impedance_real  = <real value of the impedance (if nimped < 0)>;
  impedance_imag  = <imaginary value of the impedance (if nimped < 0)>;
  imped_file      = <name of impedent panels files (if nimped > 0) >;

  nradian         = <number of radiant panels>;
  radiant_real   = <real intensity of the radiant field (if nradian < 0) >;
  radiant_imag   = <imaginary intensity of the radiant field(if nradian < 0)>;
  radiants_file  = <name of radiant panels file (if nradian > 0) >;
# <data needed if custombc=0 (or absent)- end>

# <data needed if custombc=1- begin>
  bc_lambda_file ="name of lambda file"
  bc_gamma_file = "name of gamma file"
  bc_func_file = "name of func file"
```

```
  bc_g_file = "name of vector g file"
# <data needed if custombc=1- end>

# <data needed if custombc=2- begin>
  gmshbc=["phys1", "phys2"];

  phys1={
   tag = <tag of physical group 1>;
   re_lambda = real part of λ;
   im_lambda = imaginary part of λ;
   re_gamma  = real part of γ;
   im_gamma  = imaginary part of γ;
   re_f      = real part of f;
   im_f      = imaginary part of f;
  };

  phys2={
   tag = <tag of physical group 2>;
   ...
  };
# <data needed if custombc=2- end>

 printvtk = 1 <flag to activate output in VTK format>;
 printout = 1 <flag to activate output in ASCII format>;
 printwrl = 1 <flag to activate output in VRML2 format>;
 printmsh = 1 <flag to activate output in GMSH 2.2 ASCII format>;
};
```

`sigma` and `omega` are the real and imaginary parts of the Laplace variable, $s = \sigma + j\omega$. The user can choose to provide the frequency range in radiants/second using the fields `minome` and `maxome`, or in cycles/second (Hz) with the fields `minfreq` and `maxfreq`. **Note that only `minome` and `maxome` are used if the config file includes all the four fieds**.

Frequency response analysis

`AcouSTO` allows you to perform a frequency response analysis writing in special output files (see B.3) the Total Pressure at chosen microphones and the values of the Potential at chosen control points on the surface for each one of the evaluated frequencies. In order to enable the writing of the frequency response analysis files you can add the following variables to the `modsol` section of the configuration file:

```
freq_resp_cnt=[<index of control points separated by comma>];
freq_resp_mic=[<index of microphones separated by comma>];
```

Thus, if you want to have the Total Pressure at microphones 1, 3 and 5 and the Potential at control points 10,20 and 30[1] you can modify the `modsol` section as follows:

[1] Please note that both microphone and control point indices are 0–based

```
modsol={
  ...
  freq_resp_mic=[1,3,5];
  freq_resp_cnt=[10,20,30];
}
```

APPENDIX B

I/O files structure

This is where you learn about input data format

Note that in the following, all the indices are supposed to be zero-based.

B.1 Geometry input files

B.1.1 Nodes file

NNODB lines with nodes coordinates separated by spaces, and NELMB lines containing the topology connection array `jnodb`. Note that the nodes indices stored into `jnodb` are 1-based.

```
nodes(1,1) nodes(2,1) nodes(3,1)
nodes(1,2) nodes(2,2) nodes(3,2)
nodes(1,3) nodes(2,3) nodes(3,3)
...
nodes(1,NNODB) nodes(2,NNODB) nodes(3,NNODB)
jnodb(1) jnodb(2) jnodb(3) jnodb(4)
jnodb(5) jnodb(6) jnodb(7) jnodb(8)
...
jnodb(NVELB-3) jnodb(NVELB-2) jnodb(NVELB-1) jnodb(NVELB)
```

B.1.2 Microphones file - Native `AcouSTO` format

On the first line, the dimensions of the microphones grid, then NMICS lines with the microphones coordinates separated by spaces.

```
NMICSX NMICSY NMICSZ DXMICS DYMICS DZMICS
xmics(1,1) xmics(2,1) xmics(3,1)
xmics(1,2) xmics(2,2) xmics(3,2)
xmics(1,3) xmics(2,3) xmics(3,3)
...
xmics(1,NMICS) xmics(2,NMICS) xmics(3,NMICS)
```

If the microphones are not ordered in a structured grid, then set $NMICSX = NMICS$ and $NMICSY = NMICSZ = 1$.

B.2 Acoustic environment input files

B.2.1 Sources file

NSOURC lines with the coordinates of each point source separated by spaces, followed by the real and imaginary parts of the corresponding source amplitude, separated by spaces.

```
xsour(1,1) xsour(2,1) xsour(3,1) Re[ampl(1)] Im[ampl(1)]
xsour(1,2) xsour(2,2) xsour(3,2) Re[ampl(2)] Im[ampl(2)]
...
xsour(1,NSOURC) xsour(2,NSOURC) xsour(3,NSOURC) Re[ampl(NSOURC)] Im[ampl(NSOURC)]
```

B.2.2 Standard boundary conditions

Planar waves file

`NPLANW` lines with the x,y, and z components of the wave unit vector for each plane wave in the field, separated by spaces, followed by the real and imaginary parts of the corresponding wave amplitude, separated by spaces.

```
wvec(1,1) wvec(2,1) wvec(3,1) Re[ampl(1)] Im[ampl(1)]
wvec(1,2) wvec(2,2) wvec(3,2) Re[ampl(2)] Im[ampl(2)]
...
wvec(1,NPLANW) wvec(2,NPLANW) wvec(3,NPLANW) Re[ampl(NPLANW)] Im[ampl(NPLANW)]
```

Radiant panels file

`NRADIAN` lines with the indices of the radiating panels, followed by the real and imaginary parts of the corresponding radiation

```
iradian(1) Re[ampl(1)] Im[ampl(1)]
iradian(2) Re[ampl(2)] Im[ampl(2)]
...
iradian(NRADIAN) Re[ampl(NRADIAN)] Im[ampl(NRADIAN)]
```

If `NRADIAN<0` the code reads the first line and associate the complex amplitude to the whole boundary.

Impedent panels file

`NIMPED` lines with the indices of the impedent panels, followed by the real and imaginary parts of the corresponding impedance

```
izimp(1) Re[zimp(1)] Im[zimp(1)]
izimp(2) Re[zimp(2)] Im[zimp(2)]
...
izimp(NIMPED) Re[zimp(NIMPED)] Im[zimp(NIMPED)]
```

If `NIMPED<0` the code reads the first line and associate the complex impedance to the whole boundary.

B.2.3 Custom boundary conditions

The content of the files defining the acoustic properties of the boundary changes if the geometry is given in the `AcouSTO` native format (nodes file or built–in pre–processor) or using the `Gmsh` format.

`AcouSTO` native format

The files collecting the values of the custom boundary condition functions must contain a number of lines equal to the number of panels you want to assign a value for the functions. The format of each line is

`ielmb Re[gamma(ielmb,ifreq)] Im[gamma(ielmb,ifreq)]`	for the γ file
`ielmb Re[lambda(ielmb,ifreq)] Im[lambda(ielmb,ifreq)]`	for the λ file
`ielmb Re[func(ielmb,ifreq)] Im[func(ielmb,ifreq)]`	for the $\tilde{f}$ file

If the above files provide a value for the boundary condition of only a subset of panels, the remaining ones are assumed by default to be acoustically rigid, *i.e.*, $\lambda = 1$, $\gamma = 0$, and $\tilde{f} = 0$.

`Gmsh` format

Using a `.msh` file generated by `Gmsh` , the files collecting the values of the custom boundary condition functions are generated by `AcouSTO` from the config file entries (see Sections 4.3 and A.2) using the fixed naming scheme

```
bc_lambda_file = "exported_gmshbc_lambda.dat"
```

```
bc_gamma_file = "exported_gmshbc_gamma.dat"
```

`bc_func_file = "exported_gmshbc_func.dat"`

These files are supposed to be temporary files and not stored within the generated output folder. Nevertheless, they are deliberately not removed at the end of the job, to grant to the user the possibility to save them somewhere else (foe example, to use them with an earlier version of `AcouSTO`).

Again, if the above files provide a value for the boundary condition on only a subset of the *physical groups* defined, the remaining ones are assumed by default to be acoustically rigid, *i.e.*, $\lambda = 1$, $\gamma = 0$, and $\tilde{f} = 0$.

In both cases, the **g** vector input file must include only one line (if longer, only the first record is read) containing the three components of **g**

```
g(1) g(2) g(3)
```

B.3 Output files

The output files produced by `AcouSTO` are listed in Table 4.6. Here, we provide details about their format and content. The flag `printvtk` is available to enable the output on the `*.vtk` files. The `*.out` and `*.wrl` files are always produced.

When `dirneu=1`, the output files related to the boundary contain $\partial\tilde{\varphi}/\partial n$, the normal derivative of the fucntion $\tilde{\varphi}$, since this is the function `AcouSTO` solves for in this case.

B.3.1 VRML Files

- `<RunName>-surface.wrl`

This file contains the entire geometry of the boundary. The surfaces are modeled as an `IndexedFaceSet` node without `creaseAngle`, in order to preserve the actual geometry of the panels. Default VRML RGB color is $(1, 0, 0)$.

- `<RunName>-controls.wrl`

This file contains the location of the control points on the boundary. Each point is represented by a sphere having radius $r = 0.02$. This value should be changed to fit with the overall dimensions of the boundary, when a global picture is needed (an automatic adjustment of the radius will be included in the future). Default VRML RGB color is $(0, 1, 0)$.

• `<RunName>-mics.wrl`

This file contains the location of the microphones in the field. Each point is represented by a sphere having radius $r = 0.02$. This value should be changed to fit with the overall dimensions of the boundary, when a global picture is needed (an automatic adjustment of the radius will be included in the future). Default VRML RGB color is $(0, 1, 1)$.

• `<RunName>-chief.wrl`

This file contains the location of the additional interior control points for CHIEF regularization. Each point is represented by a sphere having radius $r = 0.02$. This value should be changed to fit with the overall dimensions of the boundary, when a global picture is needed (an automatic adjustment of the radius will be included in the future). Default VRML RGB color is $(0.7, 0.7, 0.7)$.

• `<RunName>-sources.wrl`

This file contains the location of the point sources. Each point is represented by a sphere having radius $r = 0.02$. This value should be changed to fit with the overall dimensions of the boundary, when a global picture is needed (an automatic adjustment of the radius will be included in the future). Default VRML RGB color is $(0, 1, 0.4)$.

• `<RunName>-radiants.wrl`

This file contains the radiant panels. Each panel is represented as a polygon belonging to an `IndexedFaceSet` node. Default VRML RGB color is $(1, 0, 1)$.

• `<RunName>-imped.wrl`

This file contains the impedent panels. Each panel is represented as a polygon belonging to an `IndexedFaceSet` node. Default VRML RGB color is $(1, 0.5, 0)$.

B.3.2 `VTK` Files

• `<RunName>-surf-<XX.YYYY>Hz.vtk`

This file contains the solution on the boundary. The data are formatted as `DATASET POLYDATA`, associated to the panels by the nodes–vertices topology connection function. The scalar fields that can be visualized are:

1. real part, imaginary part, and absolute value of the incident field;
2. real part, imaginary part, and absolute value of the scattered field;

3. real part, imaginary part, and absolute value of the total field.

`<XX.YYYY>` is the value of the frequency in Hz. In multiple–frequency runs, one file is produced for each frequency.

- `<RunName>-mics-<XX.YYYY>Hz.vtk`

This file contains the field at the microphones array. If the microphones file is in native `AcouSTO` format, then the data are formatted as `DATASET UNSTRUCTURED_GRID`. The scalar fields that can be visualized are:

1. real part, imaginary part, and absolute value of the incident field;
2. real part, imaginary part, and absolute value of the scattered field;
3. real part, imaginary part, and absolute value of the total field;
4. total field in Db;
5. insertion loss, defined as

$$\mathcal{I}_{\mathcal{L}} = 20 \log \frac{\tilde{\varphi}_{\text{in}} + \tilde{\varphi}_{\text{sc}}}{\tilde{\varphi}_{\text{in}}}$$

`<XX.YYYY>` is the value of the frequency in Hz. In multiple–frequency runs, one file is produced for each frequency.

B.3.3 Gmsh `msh` Files

- `<RunName>-surf-<XX.YYYY>Hz.msh`

This file contains the solution on the boundary using the `Gmsh` mesh format. The scalar fields that can be visualized are:

1. real part, imaginary part, and absolute value of the incident field;
2. real part, imaginary part, and absolute value of the scattered field;
3. real part, imaginary part, and absolute value of the total field.

`<XX.YYYY>` is the value of the frequency in Hz. In multiple–frequency runs, one file is produced for each frequency.

- `<RunName>-mics-<XX.YYYY>Hz.msh`

This file contains the pressure at the microphones array using the `Gmsh` mesh format. The scalar fields that can be visualized are:

1. real part, imaginary part, and absolute value of the incident field;
2. real part, imaginary part, and absolute value of the scattered field;
3. real part, imaginary part, and absolute value of the total field;
4. total field in Db;
5. insertion loss, defined as

$$\mathcal{I}_{\mathcal{L}} = 20 \log \frac{\tilde{\varphi}_{\text{in}} + \tilde{\varphi}_{\text{sc}}}{\tilde{\varphi}_{\text{in}}}$$

`<XX.YYYY>` is the value of the frequency in Hz. In multiple–frequency runs, one file is produced for each frequency.

B.3.4 Plain text files

• `<RunName>-body.raw`

This files contains the complete set of BEM panels, in raw–data format. The raw-data format can be easily imported Blender (as well as in most of the 3D modelers). Each file record contains the coordinates of the four vertices.

```
x1   y1   z1   x2   y2   z2   x3   y3   z3   x4   y4   z4
```

The order affects the orientation of the normal vector.

• `<RunName>-surf-<XX.YYYY>Hz.out`

This file contains the solution at all the control points on the boundary, formatted in ten space–separated columns. The header of the file is formatted so as to appear as a comment when processed by `gnuplot`. The header includes all the relevant informations about the file content. Its format is

```
# Run name  = <RunName>
# Run owner = <RunOwnerName>
# Cfg file  = <ConfigFileName>
# Frequency = sigma + i*omega ( <XX.YYYY> Hz)
#
# 1    2    3    4    5       6       7       8       9       10       11      12      13
# inod xnod ynod znod re(inc) im(inc) abs(inc) re(scat) im(scat) abs(scat) re(tot) im(tot) abs(tot)
```

`<XX.YYYY>` is the value of the frequency in Hz. In multiple–frequency runs, one file is produced for each frequency.

• <RunName>-mics-<XX.YYYY>Hz.out

This file contains the solution at the microphones array, formatted in sixteen space–separated columns. The header of the file is formatted so as to appear as a comment when processed by gnuplot. The header includes all the relevant informations about the file content. Its format is

```
# Run name  = <RunName>
# Run owner = <RunOwnerName>
# Cfg file  = <ConfigFileName>
# Frequency = sigma + i*omega ( <XX.YYYY> Hz)
#
# 1    2  3  4  5       6       7       8       9       10       11      12      13
# imic xm ym zm re(inc) im(inc) abs(inc) re(scat) im(scat) abs(scat) re(tot) im(tot) abs(tot)
```

<XX.YYYY> is the value of the frequency in Hz. In multiple–frequency runs, one file is produced for each frequency.

• <RunName>-fresp-cntZZ.out

This set of files contains the frequency response at the control points selected with the freq_resp_cnt entry of the modsol block of the config file. The format is

```
# Frequency response at control point ZZ
ifreq    omega[rad/s]   freq[Hz]   re(phi)   im(phi)   abs(phi)
```

where phi is the total potential.

• <RunName>-fresp-micZZ.out

This set of files contains the frequency response at the microphones selected with the freq_resp_mic entry of the modsol block in the config file. The format is

```
# Frequency response at microphone ZZ
ifreq omega[rad/s] freq[Hz] re(inc) im(inc) abs(inc) re(scat) im(scat) abs(scat) re(tot) im(tot) abs(tot)
```

APPENDIX C

The Blender plugin

This is where we learn about AcouSTO *and* Blender

AcouSTO does not include a dedicated geometry pre-processor, and there are currently no plans to do so. For simple geometries, the generation and formatting of the data needed by the code is very simple, and can be easily accomplished with basic programming tools even by users with limited programming skills. This could not be true anymore when dealing with complex domains (AcouSTO has no limitations, in this respect!). To help the user, the AcouSTO distribution includes a simple plugin for the $3D$ creation suite Blender (http://www.blender.org), a powerful, cross–platform, open–source modeler.

The installation of the plugin is straightforward, but it changed considerably starting from Blender version 2.5.X. For this reason the distribution of AcouSTO v1.4 includes also a newer version of the plugin, compatible with Blender $\geq$ 2.5.

Please note that both the versions of the Blender plugin export the nodes so as to have unit normal vectors pointing **outside** the surface. If the user wants to perform the acoustic analysis of the enclosure bounded by the surface has to modify the nodes ordering according to the instructions in Section 3.1.

C.1 Installation on Blender $\leq$ 2.47

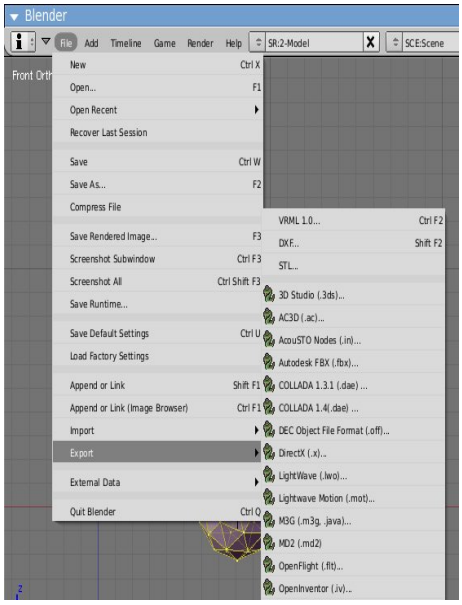


Figure C.1: AcouSTO export plugin for Blender $\leq$ 2.47.

You just need to copy the file acousto_export.py to the default directory of blender scripts (<*blender install dir*>/.blender/scripts). Under MacOSX, the directory is hidden inside the blender.app package. The script must be copied from the terminal window to the path

`/Applications/<blender dir>/blender.app/Contents/MacOS/.blender/scripts`

After you have selected the geometry you want to write in the `AcouSTO` nodes file you can export it by selecting `File->Export->AcouSTO Nodes` and the geometry will be written in `AcouSTO` format.

C.2 Installation on Blender ≥ 2.6

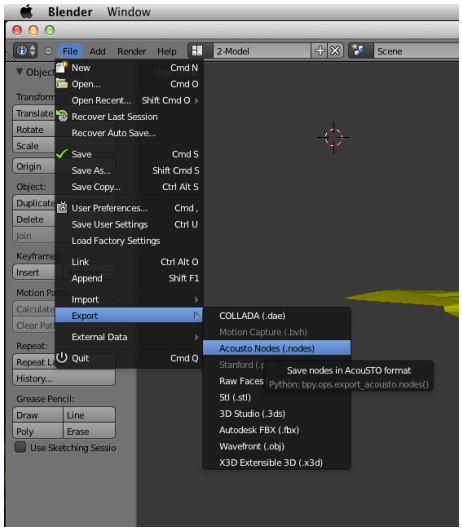


Figure C.2: `AcouSTO` export plugin for Blender ≥ 2.5.

To install the new version of the plugin, the entire folder `io_mesh_acousto` must be copied to the `scripts/addons` directory. **If you are using versions newer than 2.65, be sure to use the right script included in the distribution**. To find the exact location of the installation directory on the different operating systems, please refer to

`http://wiki.blender.org/index.php/Doc:2.6/Manual/Extensions/Python/Add-Ons`.

Once that the python scripts are in the correct place, you must activate the add-on through the Blender `File>User Preferences...` menu, under the `Addons` tab (see Fig. C.3).

To export nodes in `AcouSTO` format you have to select the geometry you are interested into, choose `File-> Export->AcouSTO Nodes` and input the file name.

Note that the newer versions of the scripts save two files:

- `FILENAME.<n.elem>.<n.vert>.nodes` - Contains the nodes and the topology. This is the file that must be provided as an input to `AcouSTO` .

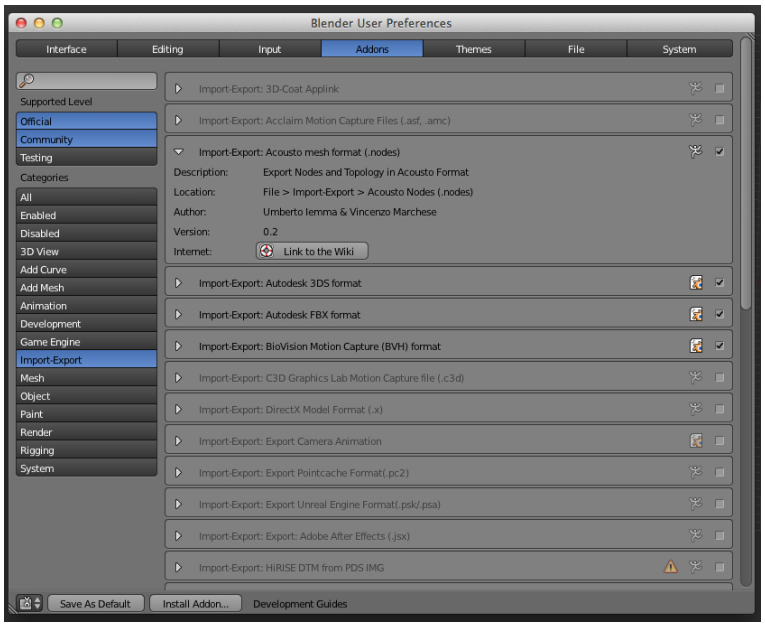


Figure C.3: Enabling the `AcouSTO` export plugin in Blender ≥ 2.5.

- `FILENAME.centroids.<n.elem>.<n.vert>.nodes` - Contains the centers of the elements. This file is not required to run `AcouSTO` but could be useful to evaluate various quantities with external codes (*e.g.*, aeroacoustic simulations).

APPENDIX D

MySQL support

This is where you can handle huge databases

D.1 Enabling MySQL support

AcouSTO can save information about its execution on a MySQL database if this feature is enabled with the option `--enable-mysql=yes` given to the `configure` script.

D.2 Database creation

AcouSTO needs a custom database to save execution information. The steps required to create this database are the following[1]:

1. create the database with the command:

   ```
   mysqladmin -u <dba user> -p create acoustorun
   ```

   where `<dba user>` is the username of the MySQL administrator

2. connect to mysql with the command

   ```
   mysql -u <dba user> -p
   ```

3. create a valid user and grant him access rights to the database with the command

   ```
   grant all on acoustorun.* to '<acousto username>'@'localhost'
                identified by '<acousto password>'
   ```

   where `<acousto username>` and `<acousto password>` are username and password of the user you want to create (without the angular brackets)

4. create the tables with the command

   ```
   mysql -U <acousto username> -p acoustorun < acousto.sql
   ```

The script `acousto.sql` is in the directory `mysql` of the distribution.

To allow AcouSTO to save on MySQL you must also add a configuration section to your configuration file as in the following:

```
mysql={
  active =<mysql flag>
```

[1] you need a MySQL administration user and password to create the database. If you don't know them, please ask your system administrator

```
  host=<mysql hostname>;
  port=<mysql port>;
  user=<acousto user>;
  password=<acousto password>;
  db=<acousto db name>;
};
```

where:

- `<mysql flag>` is 1 if you want to activate `MySQL` storing, 0 otherwise.
- `<mysql hostname>` is the name of the host where `MySQL` runs.
- `<mysql port>` is the the port `MySQL` listens to.
- `<acousto user>` is the user with access rights to the acousto database.
- `<acousto password>` is the password
- `<acousto db name>` is the name of the database (in the previous steps we called it "acoustorun"

D.3 Data storage and retrieving

`AcouSTO MySQL` features stores run information in two different tables: `runinfo` and `rundata`.

D.3.1 RUNINFO

The structure of table `runinfo` is reported in Table D.1

D.3.2 RUNDATA

The table `rundata` contains the execution results and configuration stored as BLOB (Binary Large OBject) data. These data can be retrieved and printed to the standard output in readable form by using the additional `ac_mysqlread` program that supports the following options

Column	Description
runid	primary key auto generated
categoryid	Id of execution category. This defaults to 1.
username	username of the user who has launched acousto master node process. An entry with the same username must be present in the table runuser
title	Title of execution. This is taken from the configuration file
program	Defaults to acousto
version	Version of AcouSTO
datestart	Start date of execution
dateend	End Date of execution
elapsed	elapsed time in seconds
hostname	name of host on which the process of AcouSTO master node has run
hosthos	name of OS on which the process of AcouSTO master node has run
nnodes	number of nodes of the run
notes	Additional notes (not used)

Table D.1: runinfo table structure

```
-h, --mysql-host=<hostname>    set {\tt MySQL} host (default=localhost)
-P, --mysql-port               set {\tt MySQL} port (optional)
-u, --mysql-user               set {\tt MySQL} User
-d, --mysql-dbname             set {\tt MySQL} Database name
-p, --mysql-password           set {\tt MySQL} Password
-r, --run-id                   set Run id as saved in the DB
-n, --data-name                set Name of data to be retrieved
-V, --version                  output version information
-h, --help                     this help guide
```

The valid values for the -n, –data-name option are:

- ACOUSTO_CONFIG configuration file
- ACOUSTO_BODY_TOT total potential at the control points
- ACOUSTO_BODY_INC incident potential at the control points
- ACOUSTO_BODY_SCA scattering potential at the control points
- ACOUSTO_MICS_TOT total potential at microphones
- ACOUSTO_MICS_INC incident potential at microphones
- ACOUSTO_MICS_SCA scattering potential at microphones

If you want, for instance, to save the configuration file for the run saved with id 110[2] you can issue the following command [3]:

```
ac_mysqlread -h localhost -u acousto -p acousto -r 110 -d acoustorun  \
     -n ACOUSTO_CONFIG > run110.cfg
```

If you want to save the Scattering potential at microphones for the run saved with id 110 you can issue instead the following command:

```
ac_mysqlread -h localhost -u acousto -p acousto -r 110  -d acoustorun \
     -n ACOUSTO_MICS_SCA > run110.pressure
```

the output will be a file containing the scattering potential for each calculated frequency[4].

D.4 Web frontend to MySQL (beta)

Included in the distribution there is a small php application to easily access the database once filled with data. In order to run it you must have a working LAMP (Linux-Apache-MySQL-PHP) software stack working (although Linux is not a strict requirement). You can install the web application by simply copying the content of the php folder in a folder of your choice that is under your web site document root (please refer to Apache documentation for details). Then you must change the `acousto_mysql.php` file in order to reflect your database configuration. Point your browser to the web url of the `AcouSTO` php application you configured and you should be able to see a page like the one in D.1. You can access the details of a run by clicking on the run name in the first column of the table and you will be presented a page like the one in D.2 from which you can download the configuration file and the results.

[2] if the `MySQL` feature is enabled, after completing the run the logging system of `AcouSTO` outputs the run id with the string "Run was stored with id -> ..."

[3] we assume here userid=acousto, password=acousto, dbname=acoustorun

[4] each group of values is separated by a header prepended by a # with the information on the complex frequency

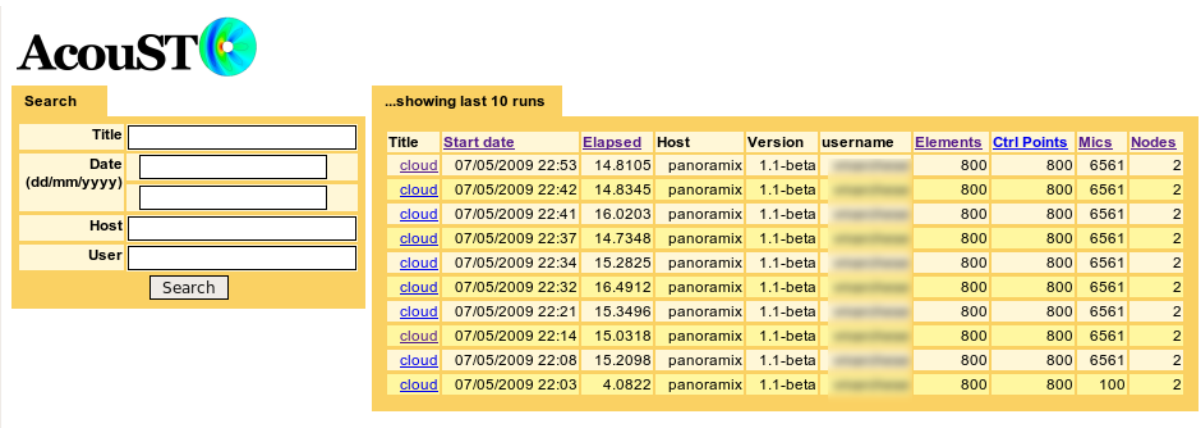


Figure D.1: Homepage of the web application

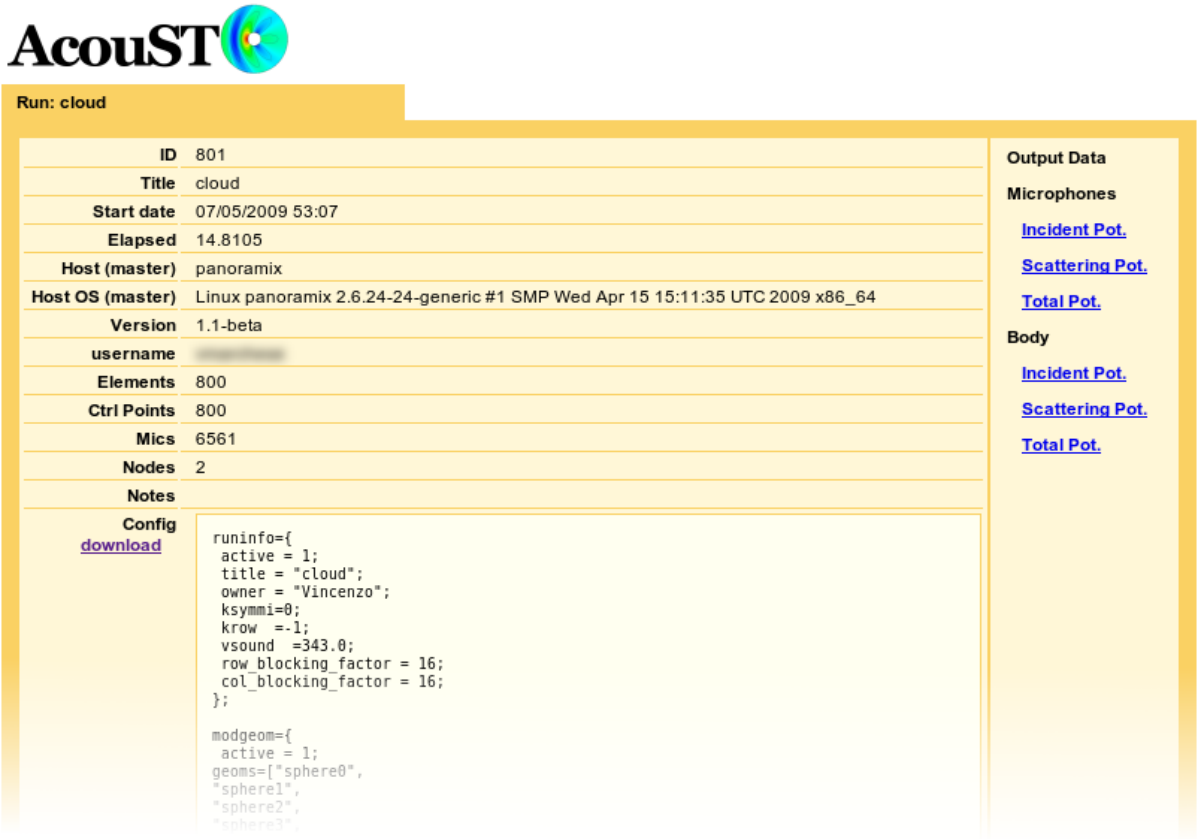


Figure D.2: Run details