

COMSOL[®] SERVER & LIVE[™]LINK[™] FOR MATLAB[®]

– GETTING STARTED GUIDE –

by Eric Ballester¹ & Théo Cavalieri²

¹ LAUM – UMR CNRS 6613, Institut d’Acoustique - Graduate School, Le Mans Université, Avenue Olivier Messiaen, 72000 Le Mans, France

² Empa, Swiss Federal Laboratories for Materials Science and Research, Acoustics/Noise control, 129 Überlandstrasse, 8600 Dübendorf, Zürich, Switzerland

Contents

1	Note of Intention	3
2	Linking COMSOL[®] to MATLAB[®]	3
2.1	COMSOL [®] Server	4
2.2	MATLAB [®] link	5
3	Case study: 2D parametric model of a quarter-wavelength resonator	6
3.1	Creating the model with COMSOL [®] GUI	6
3.1.1	From Geometry to Plotting results	6
3.2	Exporting COMSOL [®] models into MATLAB [®] files	7
3.2.1	Understanding COMSOL [®] syntax	8
3.3	Making a parametric model	15
3.3.1	Computing a problem in COMSOL [®] directly via MATLAB [®]	15
3.4	Concluding remarks	19
	Appendices	20
A	Tips and tricks	20
A.1	Add a MATLAB [®] shortcut	21
A.2	Portability over different COMSOL [®] versions	21
A.3	Compacting history	22

A.4	Fourier convention	22
A.5	Acoustic thermal and viscous dissipation models in COMSOL®	22
A.6	User validation	22
A.7	Preparation of a new model	23
A.8	Setting physical parameters	24
A.9	Saving what is necessary	25

Directory tree

```

Folder
├── Getting_Started_Guide_MATLAB-COMSOL-Livelink.pdf.....This document
├── LiveLink™ for MATLAB® User's Guide.pdf
├── readme.md
├── licence.m
├── Tutorial
│   ├── main.m
│   ├── Comsol_TubeSlit_Parametric.m
│   ├── Comsol_TubeSlit_SaveasOutput.m
│   ├── data_line.png
│   └── Models
│       └── Comsol_TubeSlit.mph

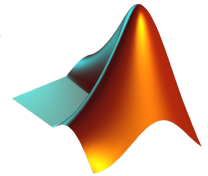
```

1 Note of Intention

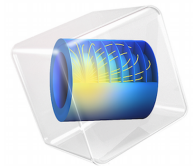
This guide is intended for people with a minimum background knowledge of both MATLAB® and COMSOL® Multiphysics software. It aims to provide a basic understanding on how to ‘link’ both software in order to be able to manipulate COMSOL® through a MATLAB® interface. Such inter-software communication can become extremely useful when designing complex systems which involve both numerical computation through MATLAB® and numerical solving through COMSOL®, e.g., systems that require computational steps between simulation iterations and/or ones whose complex geometry can be parametrically built.

In a nutshell

MATLAB® (from Matrix Laboratory) is a proprietary programming language focused on numerical computation tasks which is mostly used by engineers and scientists to analyse data and design systems. It comes with its own user-interface featuring plenty of tools to develop numerical systems and interpret MATLAB® code;



COMSOL® is a powerful multi-physics numerical solver and simulation software. The software facilitates conventional physics-based user interfaces and coupled systems of partial differential equations (PDEs). COMSOL provides an IDE and unified workflow for electrical, mechanical, fluid, acoustics, and chemical applications.



2 Linking COMSOL® to MATLAB®

In order to manipulate COMSOL® through MATLAB®, one first needs to establish a connection between the two programs. This is done via a two-step procedure:

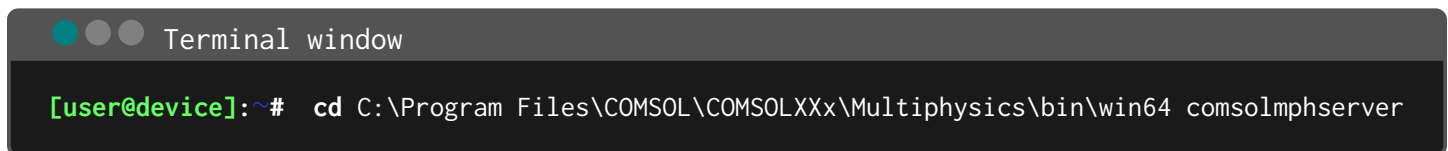
1. Launching COMSOL Server, which is a COMSOL® feature allowing COMSOL® to listen on a communication port;
2. Executing a MATLAB® command for connecting to the COMSOL Server port and sending instructions over the latter.

2.1 COMSOL® Server

Launching COMSOL® Server is usually trivial as an executable can be readily accessed in the COMSOL® installation folder. In some cases, one may want (or be required) to start COMSOL® Server from console instructions: this is detailed below.


Starting COMSOL® Server can be done by accessing the **comsol** executable file under the COMSOL® installation folder and executing it with an optional argument in a **terminal window**. The respective COMSOL® installation directory is dependent on the user's operative system (OS). The examples below show the default installation directory for Windows, Mac, and Linux OS, as well as the input command that ensues¹, where COMSOLXXx refers to the installed version of COMSOL®, e.g., COMSOL53a for COMSOL® version 5.3a.

WIN

A screenshot of a Windows terminal window titled "Terminal window". The prompt is [user@device]:~#. The command entered is cd C:\Program Files\COMSOL\COMSOLXXx\Multiphysics\bin\win64 comsolmphserver.

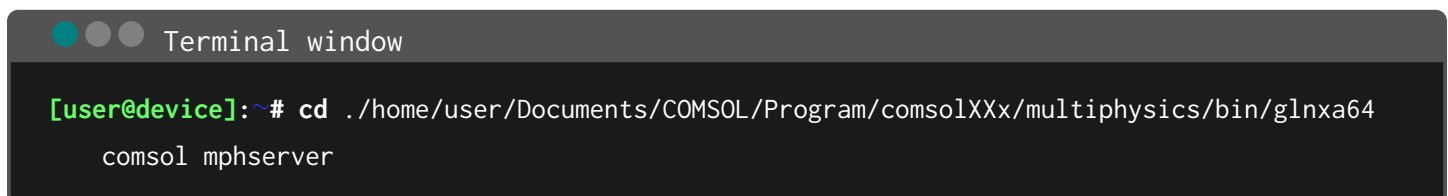
```
[user@device]:~# cd C:\Program Files\COMSOL\COMSOLXXx\Multiphysics\bin\win64 comsolmphserver
```

OSX

A screenshot of a macOS terminal window titled "Terminal window". The prompt is [user@device]:~#. The command entered is cd ./Applications/comsolXXx/Multiphysics/bin/ comsol mphserver.

```
[user@device]:~# cd ./Applications/comsolXXx/Multiphysics/bin/ comsol mphserver
```

GNU/LNX

A screenshot of a Linux terminal window titled "Terminal window". The prompt is [user@device]:~#. The command entered is cd ./home/user/Documents/COMSOL/Program/comsolXXx/multiphysics/bin/glnxa64 comsol mphserver.

```
[user@device]:~# cd ./home/user/Documents/COMSOL/Program/comsolXXx/multiphysics/bin/glnxa64 comsol mphserver
```

Once the above directory is given to the terminal, the following command will initiate COMSOL® Server and will state a listening port (e.g., 2036 by default) over which it will listen for any incoming instructions. This process of opening up COMSOL® Server can be made as a ‘one-click’ executable depending on the user's OS (see the respective documentation for each OS).

¹The bash command cd means to change directory.

2.2 MATLAB® link

In order to complete the connection between the two software, a final command needs to be entered in MATLAB® through the **command window** accessible on the main interface. This command can also be made executable by creating a custom **favourite command** (see [here](#) for further details regarding the creation of custom favourite commands, or use the example provided in the appendix [A.1](#) of this guide). Note that the [userpath] of the COMSOL® installation folder needs to be changed to the one of the OS, as seen in the previous section. **In order for this step to work, make sure that the LiveLink™ for MATLAB® module was selected during the COMSOL® installation process.**



```
MATLAB command window

1 run('[userpath]/COMSOL/Program/comsolXXx/multiphysics/mli/mphstart.m')
   % absolute path under Linux
```

A confirmation message that MATLAB® is now communicating with COMSOL® can be read in the **terminal window** used to launch COMSOL® Server, which should display the following message: A LiveLink™ for MATLAB® client with username 'username' has logged in from 'device-name'. Do not close the terminal window as this will shutdown COMSOL® Server on the particular communication port.

Following the above procedure, both COMSOL® and MATLAB® should now be linked and thus be able to communicate. If so, congratulations! Both programs are now awaiting your commands!

Summary

Launch COMSOL® Server so that a communication port can be initiated through which COMSOL® will be listening for any incoming instructions;

Start LiveLink™ for MATLAB® by calling the `mphstart.m` file which will establish a connection with COMSOL® Server through the opened communication port.

3 Case study: 2D parametric model of a quarter-wavelength resonator

This section provides a detailed example on how MATLAB® and COMSOL® can be used for parametric studies and should thus provide a minimal understanding on COMSOL® model building and on MATLAB® instructions for generating the latter in a parametric manner. In this way, we chose to model the acoustic reflection over of a quarter-wavelength resonator, i.e., a open-closed tube. We are interested in how the acoustic pressure in the duct varies with the geometry of the resonator.

3.1 Creating the model with COMSOL® GUI

The first step in creating such a model is to use COMSOL®'s graphical user interface (GUI) and build the model from scratch. This implies creating a generic geometry, attributing the different materials, implementing the physics problem with its boundary conditions, meshing, solving and finally plotting the results.

For those who desire, the complete COMSOL® model that will be used through the entirety of this guide can be found at the following [link](#). In there, the geometry of the model is built via parameters in order to help for an eventual parametrization of the model.

3.1.1 From Geometry to Plotting results

Geometry For the sake of simplicity, we will here consider a 2D model of a the quarter-wavelength resonator (QWR) placed at the end of a rectangular duct. The bottom end is rigid whereas the other end consists of an anechoic termination in Ω_{PML} with a Perfectly Matched Layer (PML), i.e., an anechoic termination [3, 8] that satisfies Sommerfeld's condition. A conceptual view can be seen in Fig. 1.

Materials & physics definitions Once the geometry of the model is built, a material is applied to all the domains; air in this case. Then, a frequency-domain physics study is chosen, with a linear fluid elastic physics domain, namely **pressure acoustics**, applied to all the domains. Initial values for all the domains are set to zero, and an incident **background pressure** field is set in Ω_{air} as a plane wave coming in the $+\mathbf{e}_x$ direction with a unitary pressure amplitude of 1 Pa.

Meshing & frequencies of interest With the geometry and physics domains set up properly, the domains can be meshed. Here, the settings will be set to **Physics-controlled mesh** with a **Finer**

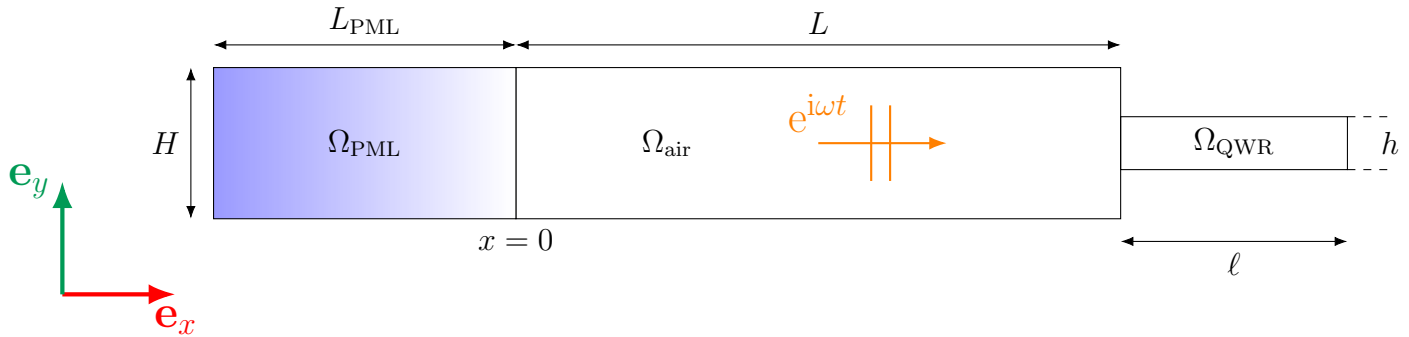


Figure 1: 2D conceptual view of a rectangular tube loaded with a QWR at one end and with a PML at the other end.

element size. The study can then be launched for any desired frequency. Due to the nature of the simulation, the frequency range considered in this study will cover acoustic waves between 5 Hz to 1000 Hz, due to the cut-off frequency of the tube being around 1500 Hz for a duct of lateral size $H = 0.2$ m.

Retrieval of physical quantities A probe is created under the **Definitions** section and placed at $(x, y)[0, 0]$, i.e., at the center of the open end of the QWR. The expression being evaluated is absolute value of the complex-valued scattered pressure field, `abs(acpr.p_s)`.

Plotting The default plotting groups created by COMSOL® will be used. These consist of `pg1`, `pg2`, and `pg3`, for the scattered pressure, total pressure level and phase of the scattered pressure at the probing point, respectively.

Once the above model is built, it can be saved as a `.mph` file (COMSOL® file format). This model will serve as a base for building similar models with MATLAB® but with different variables. In the GitHub repository of this guide, such model can be found under `./MATLAB/TubeSlit.mph`.

3.2 Exporting COMSOL® models into MATLAB® files

Once the computation of the model is completed, we can proceed to exporting the COMSOL® model into a MATLAB® file by saving the model as a `.m` file. This can be done by accessing the **File** tab, **saves as**, and choose the **Model for MATLAB .m file** as shown in Fig. 3.2. The model will be saved under the name `Comsol_TubeSlit_SaveasOutput.m`. Finally, let us switch to MATLAB® in order to view the code that was exported. It is recommended to use the **Compact History** option under the **File** tab prior to saving the model, as this will arrange the history of actions into hierarchy groups.

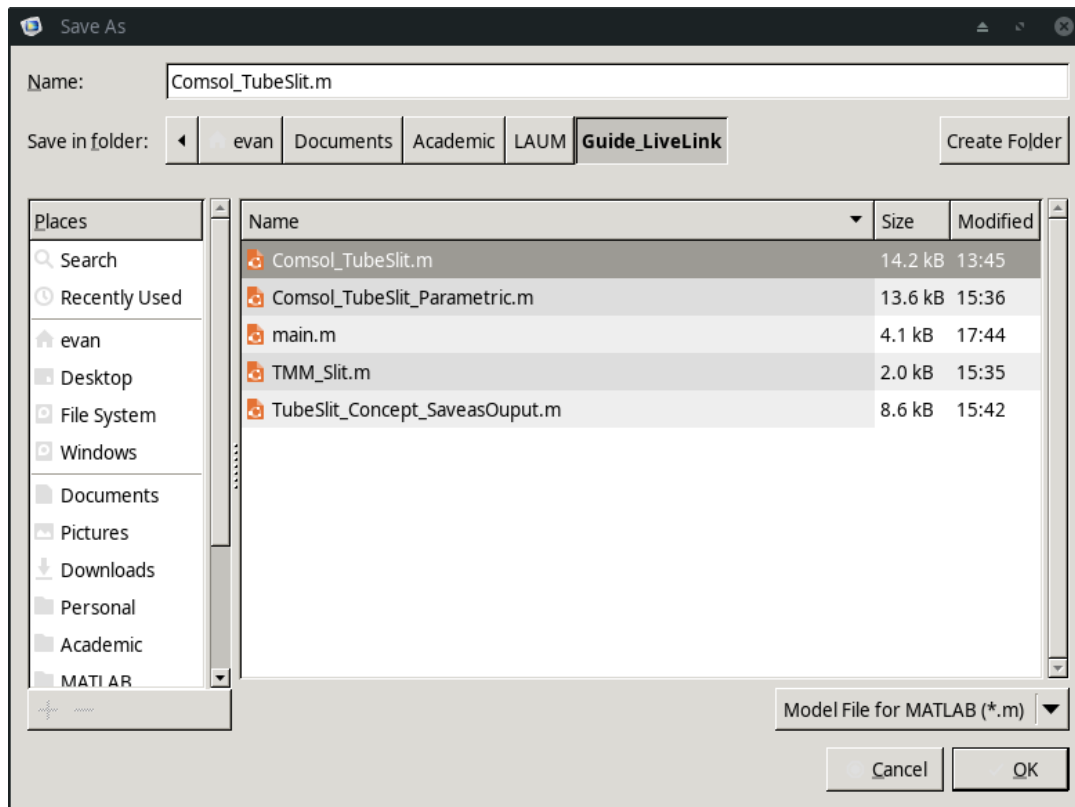


Figure 2: COMSOL® saveas option for exporting the model as a MATLAB® function file.

3.2.1 Understanding COMSOL® syntax

When looking at the MATLAB® code that was exported from COMSOL®, one can see how COMSOL® creates its model. A quick look at the code shows the necessary steps that are needed (closely corresponding to the typical COMSOL® model hierarchy):

MATLAB® function The MATLAB® code that was exported takes the form of a MATLAB® function, which for the moment shows no input arguments. These can be written at any time within round brackets right after the function name, which will be of great interest for manipulating COMSOL® through parametric variables.

Model Creation This initial step is performed in a few lines and is the same for every model that will be created. The MATLAB® code below shows a call to COMSOL® through the `model` object, with a certain absolute path, and file name and the command to launch the creation of a blank model file.


```
MATLAB command window

1 import com.comsol.model.*           %import COMSOL model lib.
2 import com.comsol.model.util.*      %import COMSOL util.
3 model = ModelUtil.create('Model');  %create model
4 model.modelPath('[absolute_path]'); %model path
5 model.label('TubeSlit_Concept.mph'); %model label
6 model.component.create('comp1', true); %create of component 1
```

Parameters Following the creation of the model, parameters needed for the model are then written. These can be recognised by the syntax `model.param[...]` which explicitly sets a value to a parameter. In the example below, a parameter `L` – the length of the slit loaded with the resonator – is assigned the value of `0.2 m`.

Note that no variable have been created in the saved COMSOL® model as none were created in the first place. However, should the need arise, analytical variables can be written in this way.

```
MATLAB command window

1 model.param.set('L', '0.3 [m]'); % set parameter L to 0.3 [m]
```

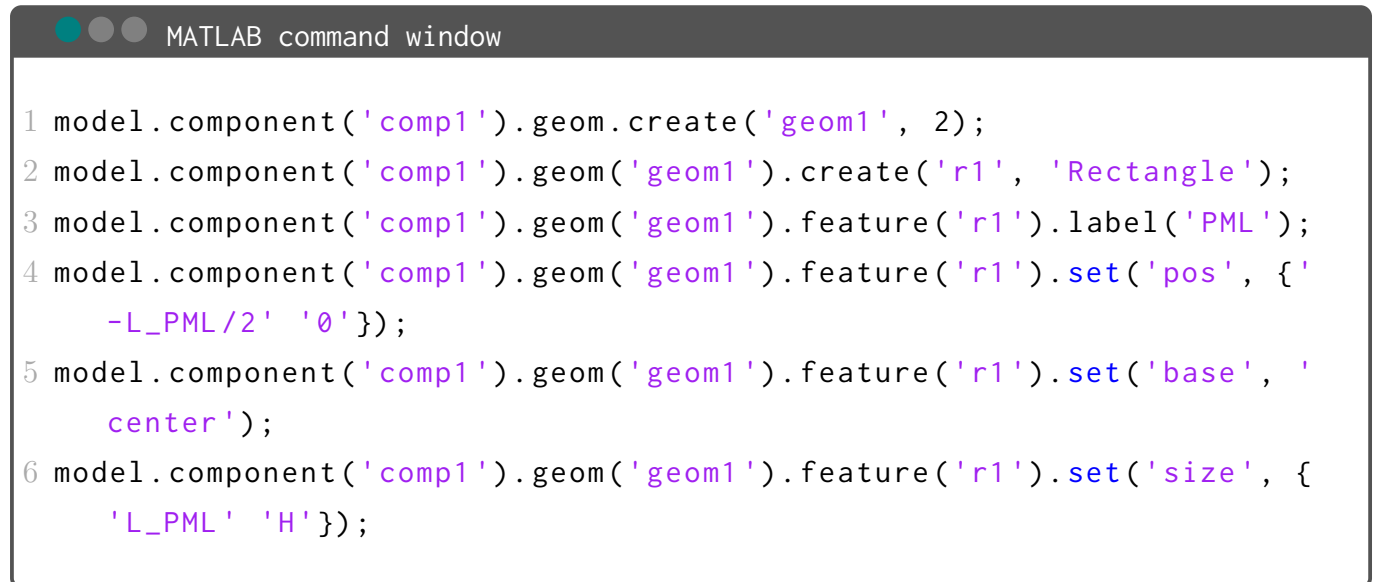
Variables Variables differ from parameters in that these can take different values. These can be written very easily. One can see, however, that a preceding command line is needed for associating variables to a variables object, here the object `var1`. Then, variables can be written with a similar syntax as parameters, i.e., `model.component.variable.set()`.

```
MATLAB command window

1 % create variable object 'var1' and set w as radial frequency
2 model.component('comp1').variable.create('var1');
3 model.component('comp1').variable('var1').set('w', '2*pi*freq');
```

Geometry In a similar way than variables, the geometry is created by first explicitly defining a geometry object, e.g., `geom1`, and the dimensional space that will be used for the model, i.e., 2 or 3 for 2D

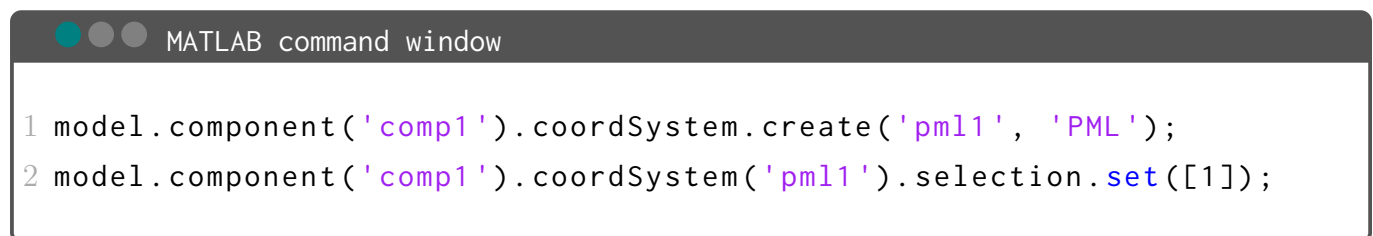
or 3D spaces, respectively. Then, any type of geometrical object can be created using the `.create` and `.feature` syntax. Below, a rectangle 'r1' is created, with two different features, viz., a position ('pos') and a size ('size'). By default the `set('base')` option is not written (for defining from which point of the geometry object the latter will be created at the defined coordinates). It can be set to 'corner' (default) or 'center' depending on the need.



```
MATLAB command window

1 model.component('comp1').geom.create('geom1', 2);
2 model.component('comp1').geom('geom1').create('r1', 'Rectangle');
3 model.component('comp1').geom('geom1').feature('r1').label('PML');
4 model.component('comp1').geom('geom1').feature('r1').set('pos', {
    -L_PML/2 '0'});
5 model.component('comp1').geom('geom1').feature('r1').set('base', '
    center');
6 model.component('comp1').geom('geom1').feature('r1').set('size', {
    'L_PML' 'H'});
```

Perfectly Matched Layer In order to satisfy the Sommerfeld condition, a PML object `pm11` is shortly defined to a domain ID, here ID #1², using the syntax below. For now, additional features are not explored and all default values will be used.



```
MATLAB command window

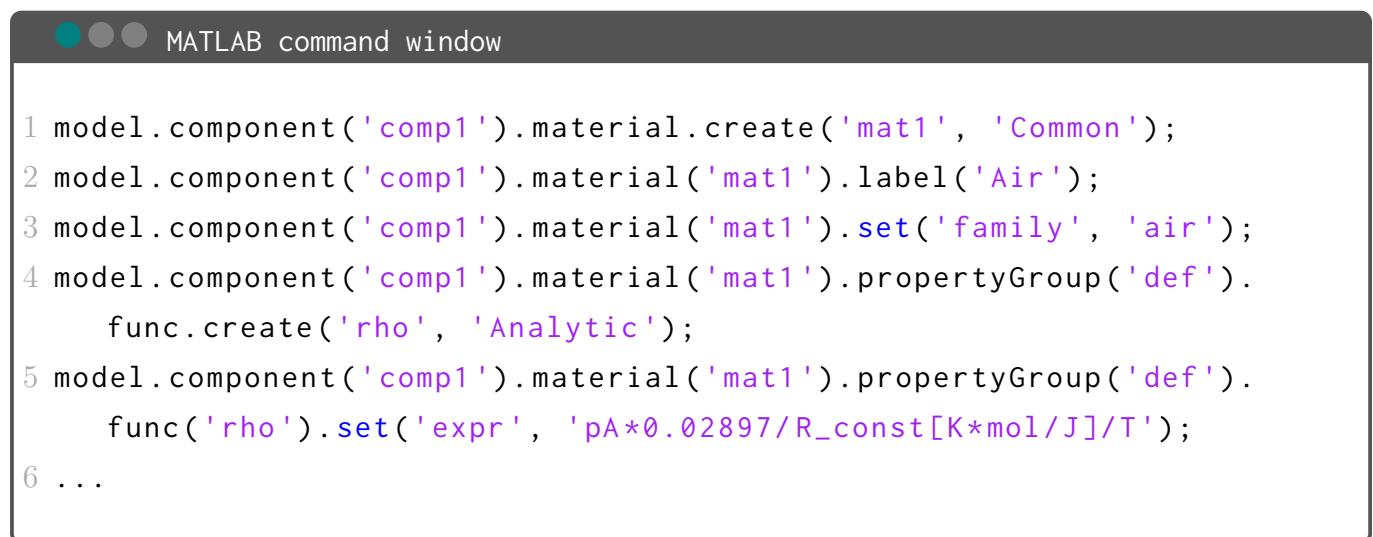
1 model.component('comp1').coordSystem.create('pm11', 'PML');
2 model.component('comp1').coordSystem('pm11').selection.set([1]);
```

Material By choosing a library material in COMSOL® during the template creation, all the different attributes of the material will be written in the code. Again, a 'mat1' material is created, with a label ('Air') and different group properties, such as density ('rho'), sound celerity ('cs'), etc. These are generally written as analytical features which are then determined with equations referring to other

²Domains are defined in numerical order from their proximity to the global origin of the model. When designing complex models with a need to track specific domain IDs, some COMSOL® functions can be used in order to obtain the ID of a domain depending on its coordinates, e.g., `mphselectbox` and `mphselectcoords`.

more fundamental parameters, such as temperature, pressure or molar gas constant, amongst others.

Understanding how a material group is defined can be helpful for creating custom materials with specific physical properties, as sometimes the numerical solving of the model only requires a few physical parameters, such as medium sound speed and medium mass density for acoustic wave propagation. These can just be directly entered as values without the need for COMSOL® internal analytical formulations.

A screenshot of a MATLAB command window titled "MATLAB command window". It contains six lines of MATLAB code. Lines 1 through 5 are indented. Line 6 is "...".

```
1 model.component('comp1').material.create('mat1', 'Common');
2 model.component('comp1').material('mat1').label('Air');
3 model.component('comp1').material('mat1').set('family', 'air');
4 model.component('comp1').material('mat1').propertyGroup('def').
    func.create('rho', 'Analytic');
5 model.component('comp1').material('mat1').propertyGroup('def').
    func('rho').set('expr', 'pA*0.02897/R_const[K*mol/J]/T');
6 ...
```

Mesh The mesh refers to the spatial discretization of the model and is responsible for the quality of the simulation, i.e., the coarser the mesh, the lower the quality but the faster the simulation. Conversely, the denser the mesh, the higher the quality but the longer the simulation time due to the increased amount of computational elements. By using the simplified menu during the creation of the model, the basic mesh-related features can be found in the exported file. We can see that the ‘mesh1’ object is created, with a ‘automatic’ mesh parameter selection enabled (‘true’), where we chose the element 3 in the rolling window (equivalent to a ‘Finer’ mesh). Then, the mesh is generated through the ‘run’ feature.

```
MATLAB command window

1 model.component('comp1').mesh.create('mesh1');
2 model.component('comp1').mesh('mesh1').automatic(true);
3 model.component('comp1').mesh('mesh1').autoMeshSize(3);
4 model.component('comp1').mesh('mesh1').run;
```

Study The study definition is fairly straightforward. A study object ‘std1’ is created with a frequency study variable (‘freq’, in Hz) which will vary along a list of values. Thus, the Frequency object can be a single number or a vector of values, as shown below.

```
MATLAB command window

1 model.study.create('std1');
2 model.study('std1').create('freq', 'Frequency');
3 model.study('std1').feature('freq').set('plist', [20 30 40 50]);
```

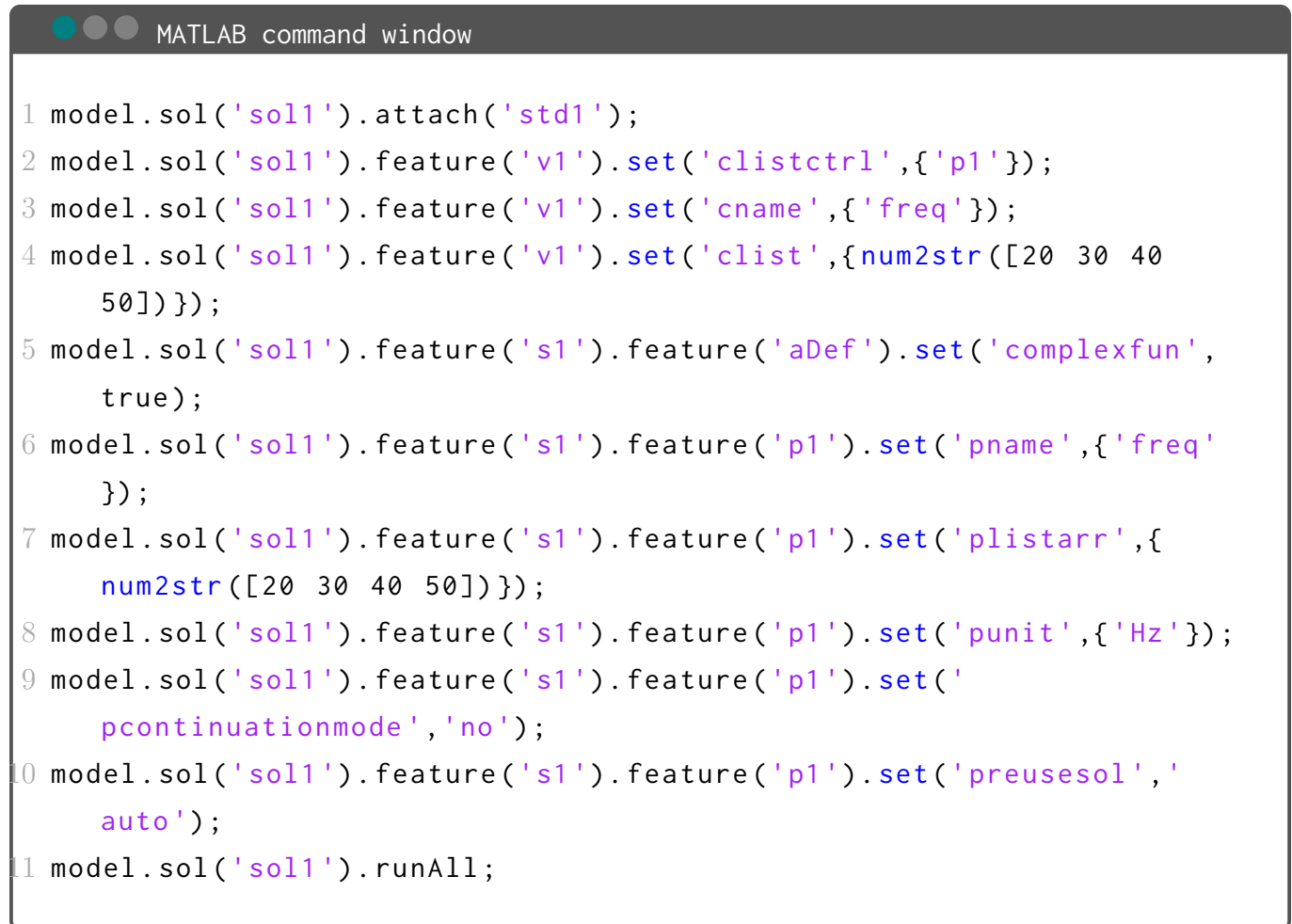
Solutions As COMSOL® solves the model, it will store the solutions within a dedicated object, here ‘sol1’, and create study steps, variables, and stationary solver objects.

```
MATLAB command window

1 model.sol.create('sol1');
2 model.sol('sol1').study('std1');
3 model.sol('sol1').attach('std1');
4 model.sol('sol1').create('st1', 'StudyStep');
5 model.sol('sol1').create('v1', 'Variables');
6 model.sol('sol1').create('s1', 'Stationary');
7 model.sol('sol1').feature('s1').create('p1', 'Parametric');
8 model.sol('sol1').feature('s1').create('fc1', 'FullyCoupled');
```

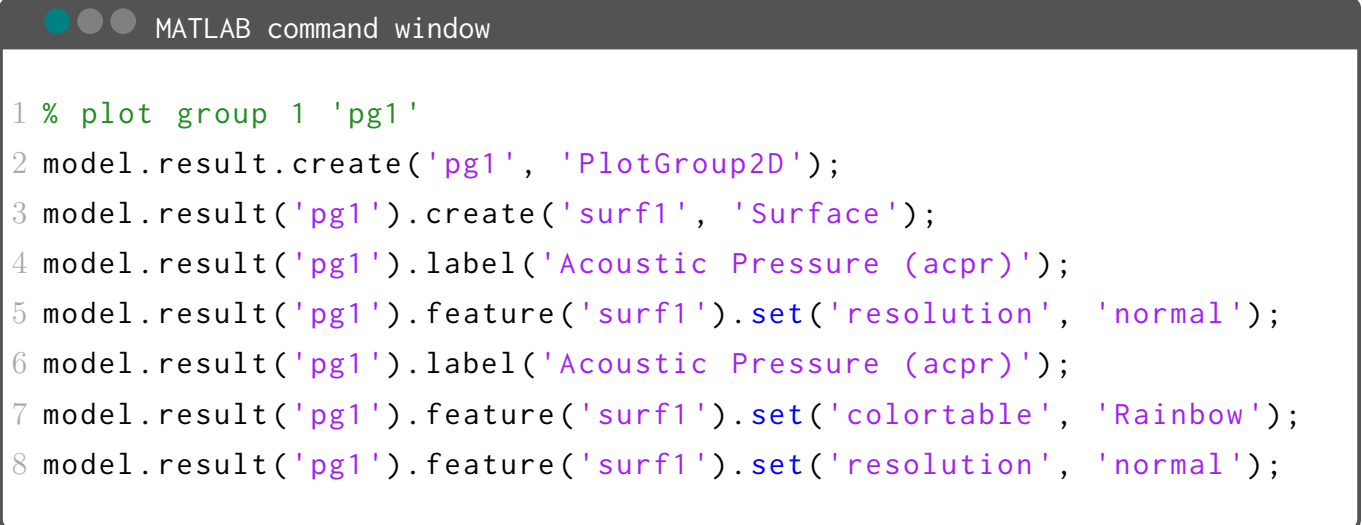
Note that the ‘clist’ feature requires the vector of frequencies to be entered in a string format, here done via the num2str() MATLAB® function. All other features, such as the complex function

toggle, or variable unit, come by default but can also be modified for any advanced uses.

A screenshot of a MATLAB command window. The title bar reads "MATLAB command window". The window contains a script with 11 lines of MATLAB code. The code is color-coded: strings are in single quotes, function names like 'set' and 'attach' are in blue, and variable names like 'model', 'sol1', 'std1', 'v1', 's1', 'aDef', 'p1', 'pname', 'plistarr', 'punit', 'pcontinuationmode', 'preusesol', and 'runAll' are in black. The code sets up a model named 'sol1', attaches a standard 'std1', sets various features like 'clistctrl', 'cname', 'clist', 'complexfun', 'pname', 'plistarr', 'punit', 'pcontinuationmode', and 'preusesol', and finally runs the model with 'runAll'.

```
1 model.sol('sol1').attach('std1');
2 model.sol('sol1').feature('v1').set('clistctrl',{'p1'});
3 model.sol('sol1').feature('v1').set('cname',{'freq'});
4 model.sol('sol1').feature('v1').set('clist',{num2str([20 30 40
    50]))});
5 model.sol('sol1').feature('s1').feature('aDef').set('complexfun',
    true);
6 model.sol('sol1').feature('s1').feature('p1').set('pname',{'freq'
    });
7 model.sol('sol1').feature('s1').feature('p1').set('plistarr',{
    num2str([20 30 40 50]))});
8 model.sol('sol1').feature('s1').feature('p1').set('punit',{'Hz'});
9 model.sol('sol1').feature('s1').feature('p1').set('
    pcontinuationmode','no');
10 model.sol('sol1').feature('s1').feature('p1').set('preusesol','
    auto');
11 model.sol('sol1').runAll;
```

Results The **results** object creates and stores the solutions of the acoustic problem. The user can then access this solution either by exporting and/or by plotting parts or the entirety of the result. This includes such as total acoustic pressure, scattered pressure, and sound pressure levels. Below, a surface plot group (**pg1**) is created. It is given a label, an expression to evaluate, a unit and a resolution, and other custom features.

A screenshot of a MATLAB command window titled "MATLAB command window". It contains eight lines of MATLAB code for creating a plot group. The code is as follows:

```
1 % plot group 1 'pg1'
2 model.result.create('pg1', 'PlotGroup2D');
3 model.result('pg1').create('surf1', 'Surface');
4 model.result('pg1').label('Acoustic Pressure (acpr)');
5 model.result('pg1').feature('surf1').set('resolution', 'normal');
6 model.result('pg1').label('Acoustic Pressure (acpr)');
7 model.result('pg1').feature('surf1').set('colortable', 'Rainbow');
8 model.result('pg1').feature('surf1').set('resolution', 'normal');
```

This plot groups can be brought as a MATLAB® figure through the COMSOL® `mphplot()` function, as written below.

A screenshot of a MATLAB command window titled "MATLAB command window". It contains two lines of MATLAB code for plotting a figure. The code is as follows:

```
1 figure;
2 mphplot(model, 'pg1');
```

Data probing The data probing done through the domain point probe is recorded through a table, `tb1`. So far, the only way the exported code allows us to access its values is through an eventual plotting of its plot group, `pg3`. However, sometimes one may want to directly evaluate a physical quantity directly while COMSOL® is running, such as the phase of the scattered pressure, in order to process it further in MATLAB®. This can be achieved through the use of COMSOL® functions, dependent on the type of the probe (e.g., point, surface, volume). This particularly useful aspect is explored in more details in the next section involving the creation of a MATLAB® function, the purpose of which being to run a COMSOL® model with parametric inputs and obtaining the respective desired output data for further data analysis.

Advanced users can also directly retrieve all information about the mesh with `mphxmeshinfo()`, `mphmeshstats()`, and elementary matrices of the FE problem using `mphmatrix()`

Hierarchy Building It is recommended, as good practice, to organise the output MATLAB® file ex-

ported through COMSOL® based on its model hierarchy. An example is provided in Appendix A.7. The file `Cmsol_TubeSlit.m` in the GitHub folder `./MATLAB/` follows such organisation.

3.3 Making a parametric model

Now that the MATLAB® file exported through COMSOL® has been organised, it can be run as is in the MATLAB® command window.



```
MATLAB command window

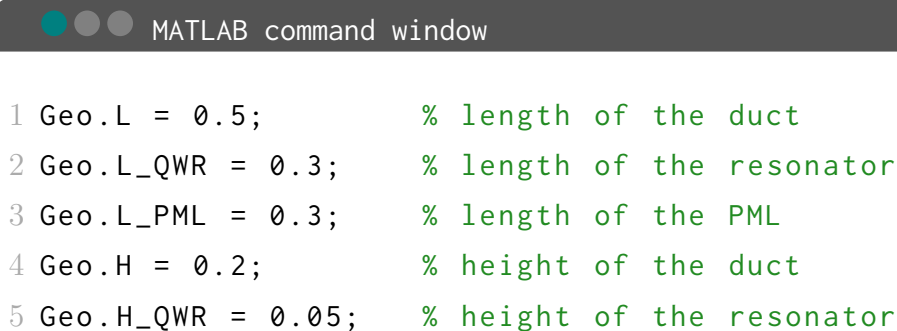
1 [out] = Cmsol_TubeSlit()
```

This function effectively builds and runs the COMSOL® model, outputting the desired figures. Yet, access to COMSOL®'s simulation data is not available yet, in addition to the variables of the model being fixed in the file.

Now, it would be of interest to give some input variables to the COMSOL® function so that different models and different output values can be given out. To this end, a MATLAB® script has to be created, in which the variables of the problem will be defined. In the GitHub repository `./MATLAB/`, a `main.m` file addressing such endeavour is accessible.

3.3.1 Computing a problem in COMSOL® directly via MATLAB®

In short, the `main.m` file defines the geometry variables of the problem, such as the length L of the QWR, the length L_{PML} of the PML, and the height H for both domains.



```
MATLAB command window

1 Geo.L = 0.5;           % length of the duct
2 Geo.L_QWR = 0.3;       % length of the resonator
3 Geo.L_PML = 0.3;       % length of the PML
4 Geo.H = 0.2;           % height of the duct
5 Geo.H_QWR = 0.05;      % height of the resonator
```

It also defines frequency-related information, such as frequency and radial frequency vectors spanning

from a lower bound to an upper bound through some defined step values.

```
MATLAB command window

1 Freq.fmin = 5;                % Minimum frequency of interest
2 Freq.fmax = 1000;            % Maximum frequency of interest
3 Freq.Df = 2;                 % Frequency step
4 Freq.Vector = (Freq.fmin:Freq.Df:Freq.fmax); % Frequency vector
5 Freq.Nf = numel(Freq.Vector); % # of frequencies
6 Freq.OmegaVector = 2.*pi.*Freq.Vector; % Radial Freq vector
```

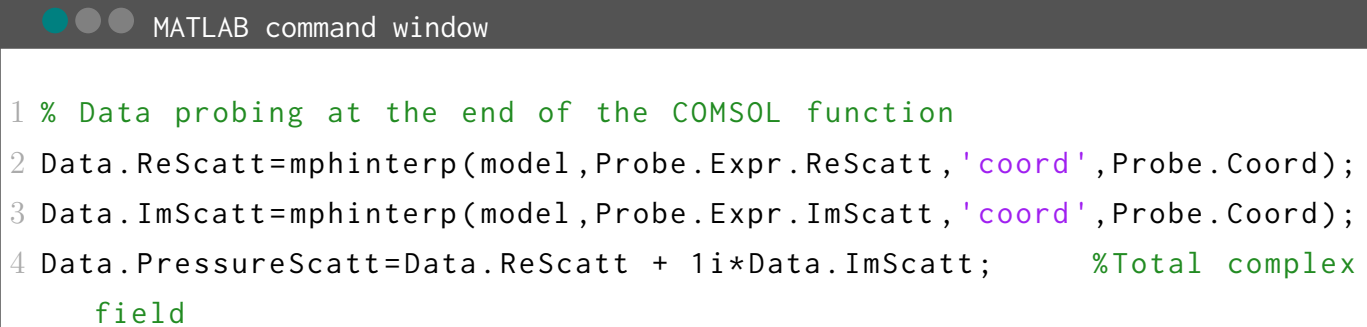
Concerning the COMSOL® model in MATLAB®, a **File** and **Probe** structures are created, containing information such as the file name that will be saved at the end of the simulation (with its extension and path), the coordinate of the probe that will be placed, as well as the expression that will be evaluated. Note that the real and imaginary parts will be defined separately as COMSOL® separates both values.

```
MATLAB command window

1 %COMSOL FILE INFORMATION
2 File.Path = [pwd,'Models'];
3 File.Tag = 'Comsol_TubeSlit';
4 File.Extension = '.mph';
5 %COMSOL PROBE INFORMATION
6 Probe.Exp.Real = 'real(acpr.p_t)';
7 Probe.Exp.Imag = 'imag(acpr.p_t)';
8 Probe.CoordPoint(1,:) = Geo.L;
9 Probe.CoordPoint(2,:) = (Geo.H + Geo.H_QWR)/2;
10 Probe.Resolution = 1000;
11 Probe.CoordLine(1,:) = linspace(-Geo.L_PML,Geo.L+Geo.L_QWR,Probe.
    Resolution);
12 Probe.CoordLine(2,:) = Geo.H/2*ones(1,Probe.Resolution);
```

At this point, the COMSOL® function file needs to be amended in function of the variables that have

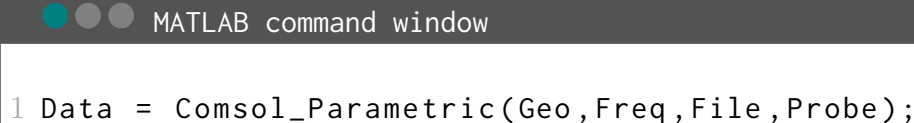
been created in MATLAB®. The file `Comsol_TubeSlit_Parametric.m` shows such process. Additionally, a probe element has been added after the study step in order to get the data out of the probe into a MATLAB® structure named `Data`. This makes use of internal COMSOL® functions, such as `mphinterp` in order to interpolate data of the given expression at the desired coordinates. More information about COMSOL® functions can be found in the official LiveLink™ for MATLAB® guide made by COMSOL® (also uploaded in the GitHub repository).



```
MATLAB command window

1 % Data probing at the end of the COMSOL function
2 Data.ReScatt=mphinterp(model,Probe.Expr.ReScatt,'coord',Probe.Coord);
3 Data.ImScatt=mphinterp(model,Probe.Expr.ImScatt,'coord',Probe.Coord);
4 Data.PressureScatt=Data.ReScatt + 1i*Data.ImScatt;      %Total complex
    field
```

Once the COMSOL® function has been arranged as in `Comsol_Parametric.m`, one can compute the COMSOL® function with the following inputs:



```
MATLAB command window

1 Data = Comsol_Parametric(Geo,Freq,File,Probe);
```

This builds and runs the simulation in function of the new input arguments made in the main MATLAB® script. It outputs the `Data` structure with the real and imaginary part defined for the `Probe` input structure. Figures 3 and 4 show the solution for acoustic pressure at a point $\mathcal{P} = \{(x, y) \in \mathbb{R}^2 \mid x = 0, y = H/2\}$ and along the line $\mathcal{L} = \{(x, y) \in \mathbb{R}^2 \mid y = H/2\}$ respectively. On Fig. 3 the modulus (a) and phase (b) of the absolute pressure are shown with respect to frequency.

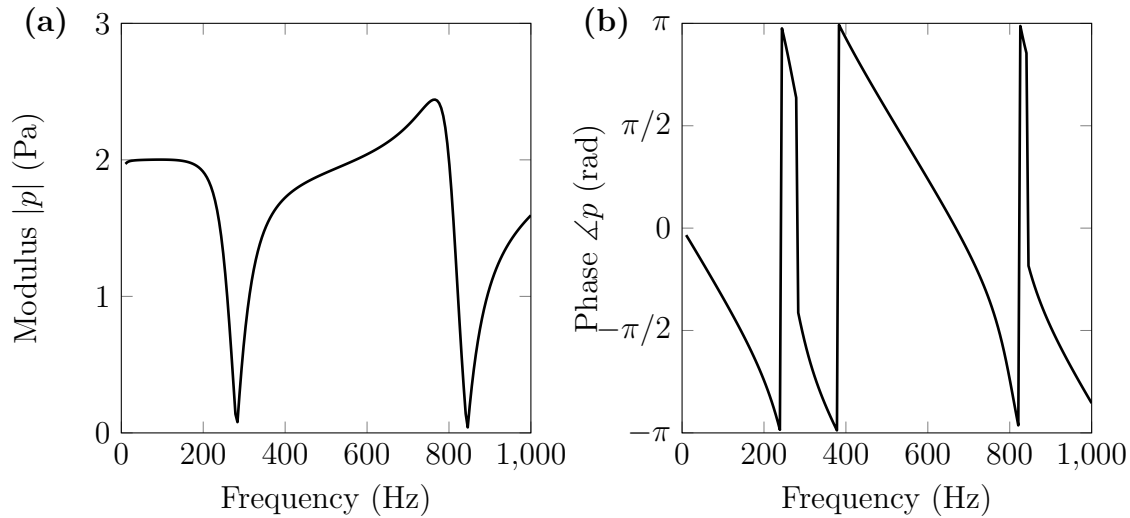


Figure 3: Modulus **(a)** and phase **(b)** of the total acoustic pressure at the position \mathcal{P} .

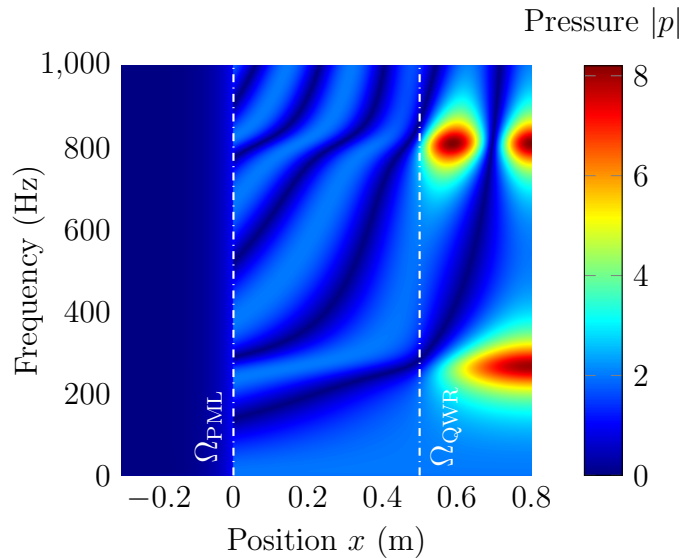


Figure 4: Modulus of the total acoustic pressure along \mathcal{L} as a function of frequency and position.

On the other hand, the modulus of the acoustic pressure is shown in a position/frequency map along \mathcal{L} in Fig. 4. We observe the quarter-wavelength modes at $f \approx 280$ Hz and $f \approx 850$ Hz, which are related to the troughs and phase jumps in Fig. 3. In addition, it is observed that the PML is correctly implemented as the pressure amplitude decreases strongly within Ω_{PML} .

3.4 Concluding remarks

It is hoped that this guide can help those who are eager to start using COMSOL® and MATLAB® hand in-hand in order to accelerate their working framework. At this point, it is suggested, as further practice, to try new situations, new models (optimization routines for example!), in order to further develop an understanding of the LiveLink™ tool. Sometimes, the process can seem tedious, but with patience and research (into Forums, the LiveLink™ guide, etc.) one will significantly deepen its skills using both interfaces. Practice makes perfect.

Should you value this work and want to share it around or cite it, the authors would like to thank you for this action. The following appendices provide some tips and tricks which may help you avoid commonly-encountered mistakes and improve your parametric models!

This work is licensed under a [Creative Commons “Attribution-NonCommercial-ShareAlike 3.0 Unported”](#) license.

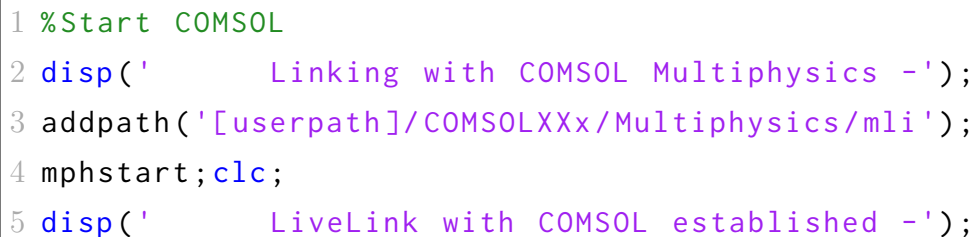


Appendices

Appendix A Tips and tricks

A.1 Add a MATLAB® shortcut

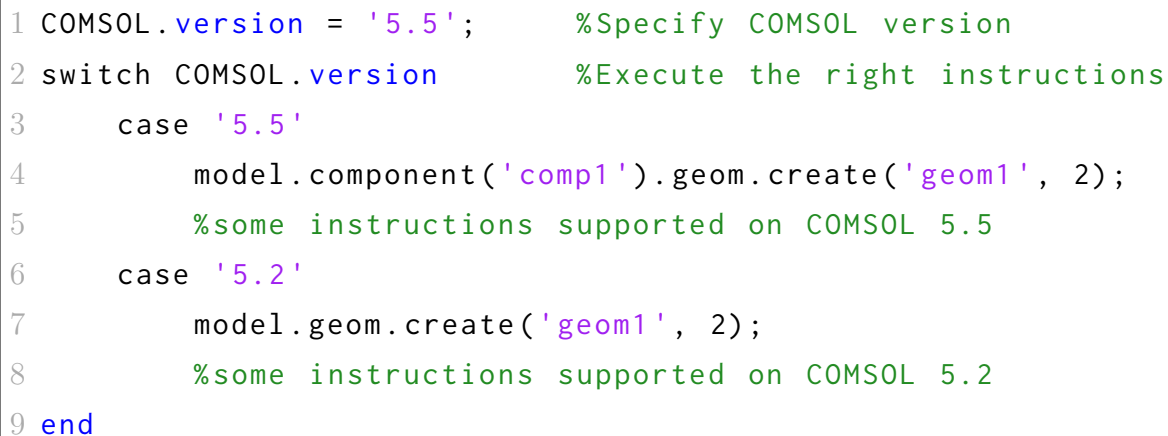
Adding this set of commands as a MATLAB® favourite command creates a shortcut, allowing the user to link with COMSOL® directly from MATLAB® quick access bar. Here again [userpath] needs to be explicitly specified according to the user's operating system.



```
1 %Start COMSOL
2 disp('      Linking with COMSOL Multiphysics -');
3 addpath('[userpath]/COMSOLXXx/Multiphysics/mli');
4 mphstart;clc;
5 disp('      LiveLink with COMSOL established -');
```

A.2 Portability over different COMSOL® versions

While the overall instructions used by COMSOL® are similar from one version to another, some differences can be noticed. In order to create a script/model which is supported on multiple versions, we recommend to use a switch. In this way, the same MATLAB® script may be executed over different versions of the COMSOL® server.



```
1 COMSOL.version = '5.5';      %Specify COMSOL version
2 switch COMSOL.version      %Execute the right instructions
3     case '5.5'
4         model.component('comp1').geom.create('geom1', 2);
5         %some instructions supported on COMSOL 5.5
6     case '5.2'
7         model.geom.create('geom1', 2);
8         %some instructions supported on COMSOL 5.2
9 end
```

A.3 Compacting history

Saving a COMSOL® model as a .m file creates a series of instructions in a chronological order. While this can appear to be useful to track all the changes that have been made to a particular model, or to find out the right instructions to implement in a model, this can make the .m file very long and tedious to navigate through. A simple alternative allows COMSOL® to only export the ‘effective’ commands, i.e., all unnecessary and overwritten instructions are removed. This is performed under the COMSOL® GUI by clicking **Compact History** in the **File** drop-down menu before saving the model as a .m file.

A.4 Fourier convention

Please note that COMSOL® uses time convention $e^{+i\omega t}$. This may come in conflict with user-defined complex physical quantities, i.e., wave-number, specific impedance, or effective properties of a porous material. If needed, remember to conjugate these quantities using the internal function `conj()`.

A.5 Acoustic thermal and viscous dissipation models in COMSOL®

The volume property **Narrow Region Acoustics** provides a built-in set of well-known models for the properties of equivalent fluids, i.e., Delany & Bazley (REF), Stinson [9], JCAL [5], and so on. More importantly, this allows the user to input custom effective properties, namely an impedance Z , a wave-number k , a bulk modulus B , and a mass density ρ . These quantities can be complex and frequency-dependent and have to be defined in the **variables** section.

A.6 User validation

In some cases where the geometric parameters are unbounded, some domains can overlap and create conflicts. Displaying the geometry and prompting the user whether it is valid is performed by the following commands:

```
MATLAB command window

1 model.geom('geom1').run;           %Build entire geometry
2 if ListGeom.doPlot                 %Plot geometry?
3     figure;mphgeom(model);         %Display geometry
4     kk = input('RUN this model?'); %Waits for user approval
5     clear kk
6 end
```

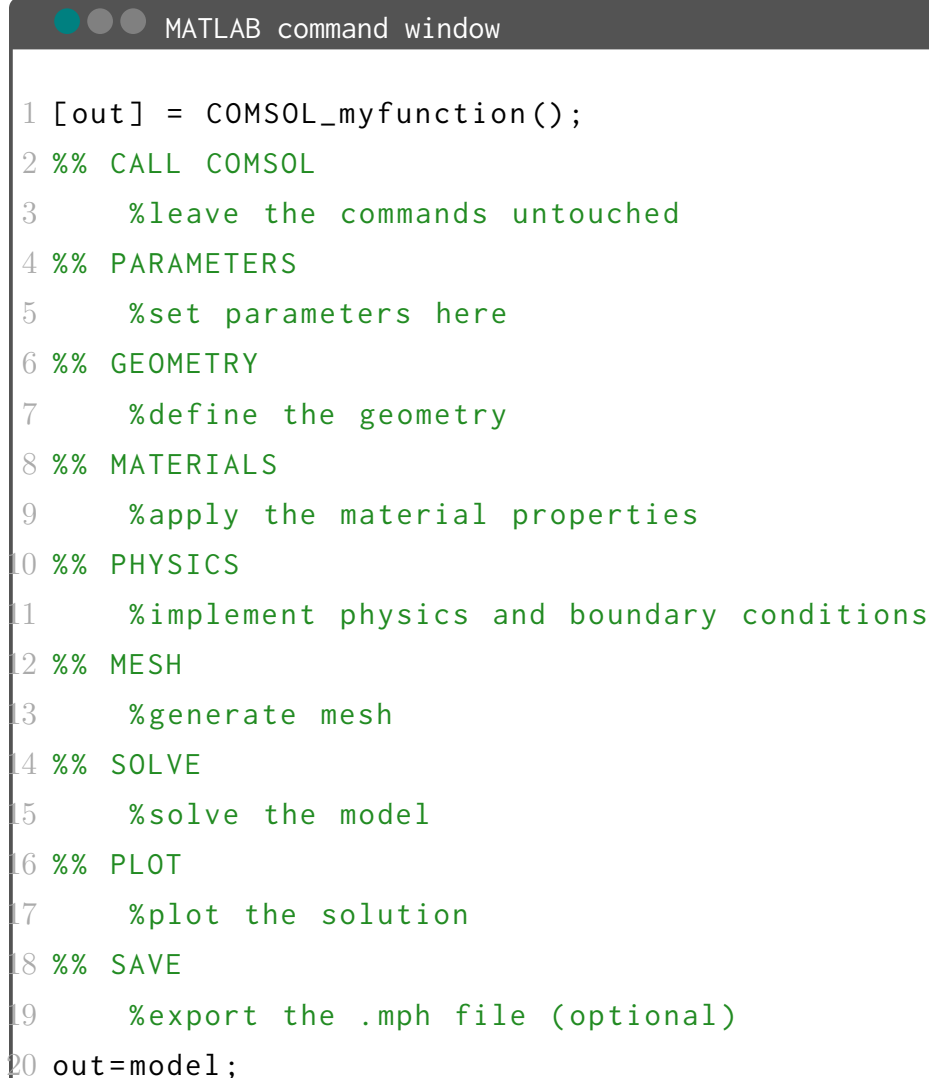
An other quite relevant criterion is the mesh quality which is displayed to the user as follows:

```
MATLAB command window

1 m_info = mphmeshstats(model,'mesh1'); %Get the mesh info
2 %Display mesh info in command window
3 disp(['Mesh has ',num2str(round(sum(m_info.numelem)/1e3)), ' kElem']);
4 disp(['Mesh mean quality is ',num2str(100*m_info.meanquality),' %']);
5 if Mesh.doPlot                     %Plot the mesh?
6     figure;mphmesh(model,'mesh1'); %Display mesh
7     kk = input('Continue?');       %Waits for user approval
8     clear kk
9 end
```

A.7 Preparation of a new model

By default, when using the `saveas` function on the model, COMSOL® will export all instructions in a chronological order. The list of commands can be compacted, i.e., unnecessary instructions are removed, using `File>Compact history` in COMSOL®'s drop-down menu. After saving the whole model as a `.m` MATLAB® function, it is strongly recommended for the user to re-organise all the instructions making use of sections.



```
MATLAB command window

1 [out] = COMSOL_myfunction();
2 %% CALL COMSOL
3     %leave the commands untouched
4 %% PARAMETERS
5     %set parameters here
6 %% GEOMETRY
7     %define the geometry
8 %% MATERIALS
9     %apply the material properties
10 %% PHYSICS
11     %implement physics and boundary conditions
12 %% MESH
13     %generate mesh
14 %% SOLVE
15     %solve the model
16 %% PLOT
17     %plot the solution
18 %% SAVE
19     %export the .mph file (optional)
20 out=model;
```

A.8 Setting physical parameters

While it is possible for COMSOL® to read the air properties from the structure of strings `ListAirPropStr`, one can also directly create a set of parameters. In this way, COMSOL® expressions can use `Rho0` instead of `ListAirPropStr.Rho0`, which improves visibility when building equations and debugging.


```
MATLAB command window

1 %-----%
2 %% COMSOL PHYSICAL CONSTANTS
3 disp('-- Set global constants')
4 %-----%
5 listfields = fieldnames(ListAirPropStr);    %Strings
6 nb_constants = numel(listfields);           %Number of constants
7 for ii = 1:nb_constants                     %Set parameters in COMSOL
8     model.param.set(listfields(ii),num2str(eval(['ListAirPropStr.',
9         char(listfields(ii))]))); %Create COMSOL parameters
9 end
```

A.9 Saving what is necessary

After having performed a parametric study, you may or may not want to save the mesh of the FEM model, as well as the resulting solutions. A COMSOL® model is actually quite light and only occupies few kilobytes of space, whereas the mesh information and solution vectors may take up to several gigabytes. By default, while using the command `mphsave()`, everything is stored. The following instructions allow to export the COMSOL® model as `.mph` file without the mesh and the solutions.

```
MATLAB command window

1 model.sol('sol1').clearSolutionData;    %Clear solution data
2 model.mesh.clearMeshes;                 %Clear meshes
3 mphsave(model, mypath);                  %Save COMSOL model
```

References

- [1] COMSOL multiphysics®.
- [2] J.-F. Allard and N. Atalla. *Propagation of sound in porous media: modelling sound absorbing materials*. Wiley, 2nd ed edition.
- [3] J.-P. Berenger. A perfectly matched layer for the absorption of electromagnetic waves. 114(2):185–200, 1994-10.
- [4] D. L. Johnson, J. Koplik, and R. Dashen. Theory of dynamic permeability and tortuosity in fluid-saturated porous media. 176(-1):379, 1987-03.
- [5] D. Lafarge, P. Lemarinier, J. F. Allard, and V. Tarnow. Dynamic compressibility of air in porous structures at audible frequencies. 102(4):1995–2006, 1997-10.
- [6] M. C. Pease. *Methods of Matrix Algebra*. Elsevier NetLibrary, Incorporated [distributor, 1965. OCLC: 944502142.
- [7] W. H. Press. *Numerical recipes: the art of scientific computing*. Cambridge University Press, 3rd ed edition, 2007. OCLC: ocn123285342.
- [8] A. Sommerfeld. *Partial differential equations in physics*. Number by Arnold Sommerfeld ; 6 in Lectures on theoretical physics. Acad. Pr, 1964. OCLC: 250848097.
- [9] M. R. Stinson. The propagation of plane sound waves in narrow and wide circular tubes, and generalization to uniform tubes of arbitrary cross-sectional shape. 89(2):550–558, 1991-02.

Acknowledgements

The authors are thankful to their respective institution for the technical and administrative support. Thanks to Dr. Logan Schwan for his precious insight on linking MATLAB® with COMSOL®, without which the present guide would not exist.



Empa

Materials Science and Technology

Contact information:

- Eric Ballestero: [eric.ballestero\[at\]univ-lemans\[dot\]fr](mailto:eric.ballestero@univ-lemans.fr)
- Théo Cavalieri: [tcaval\[at\]univ-lemans\[dot\]fr](mailto:tcaval@univ-lemans.fr) / [theo.cavalieri\[at\]empa\[dot\]ch](mailto:theo.cavalieri@empa.ch)

This work is licensed under a [Creative Commons “Attribution-NonCommercial-ShareAlike 3.0 Unported”](https://creativecommons.org/licenses/by-nc-sa/3.0/) license.

