MultiFEBE (version 2.0.0) Reference Manual

Jacob D.R. Bordón

Guillermo M. Álamo

Luis A. Padrón

Juan J. Aznárez

Orlando Maeso

July 2022

Copyright Notice and Disclaimers

Acknowledgments

This software has been developed with the support of research projects:

- PID2020-120102RB-I00, funded by the Agencial Estatal de Investigación of Spain, MCIN/AEI/10.13039/501100011033.

- ProID2020010025, funded by Consejería de Economía, Conocimiento y Empleo (Agencia Canaria de la Investigación, Innovación y Sociedad de la Información) of the Gobierno de Canarias and FEDER.

- BIA2017-88770-R, funded by Subdirección General de Proyectos de Investigación of the Ministerio de Economía y Competitividad (MINECO) of Spain and FEDER.

Contents

Chapter 1

Overview

In this chapter, a brief overview of MultiFEBE download, install and usage is given. Since MultiFEBE is open source (GPLv2 license), it can also be compiled by you, see Appendix A.

1.1 Introduction

MultiFEBE is a solver for performing linear elastic static and time harmonic analyses of continuum and structural mechanics problems comprising multiple interacting regions. It is based on the Finite Element Method (FEM) and the Boundary Element Method (BEM), which are combined in order to offer many advanced features not found elsewhere. These distinctive features have been published in many scientific papers, from which [1, 2, 3, 4, 5] can be highlighted. No doubt, this program is heir to boundary element codes published by Professor Jose Domínguez [6].

Linear elastic solid materials are available for regions modeled by finite elements or boundary elements in static and time harmonic analysis, whereas linear poroelastic (Biot's theory) and inviscid fluid materials are available only for regions modeled by boundary elements in time harmonic analysis (wave propagation problems).

It has a full set of linear elastic finite elements:

- Solid (or continuum) finite elements:
  - For two-dimensional plane strain / plane stress problems:
    * 3 or 6 nodes triangular element.
    * 4, 8 or 9 nodes quadrilateral element.
- Structural finite elements:
  - 2D/3D bar element (2 nodes).
  - 2D/3D beam elements (doubly symmetric cross-section):
    * 2 or 3 nodes Euler-Bernoulli or Timoshenko straight element.
    * 3 or 4 nodes curved element based on the degeneration from solid (Timoshenko theory).
  - 3D shell elements (Reissner-Mindlin theory):
    * 3 or 6 nodes triangular shell element based on the degeneration from solid (full/selective/reduced integration).
    * 4, 8 or 6 nodes quadrilateral shell element based on the degeneration from solid (full/selective/reduced integration).
    * 9 nodes quadrilateral MITC element (locking-free).
- Discrete finite elements:
  - 2D/3D discrete translational and translational-rotational spring/dashpot 2 node elements.
  - 2D/3D mass elements.

It also has a full set of boundary elements for linear elastic solids, poroelastic media and inviscid fluids:

- For two-dimensional plane strain / plane stress problems:

  - 2, 3 or 4 line ordinary or crack boundary elements.
  - 2, 3 or 4 line body load elements.

- For three-dimensional problems:
  - 3 or 6 triangular ordinary or crack boundary elements.
  - 4, 8 or 9 quadrilateral ordinary or crack boundary elements.
  - 2, 3 or 4 line body load elements (only in elastic regions).
  - 3 or 6 triangular body load elements.
  - 4, 8 or 9 quadrilateral body load elements.

Finite elements can be coupled to ordinary boundary, crack boundary and body load elements in order to study Soil-Structure, Fluid-Structure and Soil-Fluid-Structure Interaction in wave propagation problems. This type of mcoupling

Figure 1.1 shows an example of the modeling capabilities, where an Offshore Wind Turbine installed on a jacket structure founded on suction caissons. Finite elements model the whole foundation and structure, which includes additional masses in order to incorporate Water-Structure Interaction, and boundary and body load elements (geometrically coincident with suction caissons skirts and lids) for modeling Soil-Structure Interaction. Soil free-surface discretization is not shown for the sake of clarity.

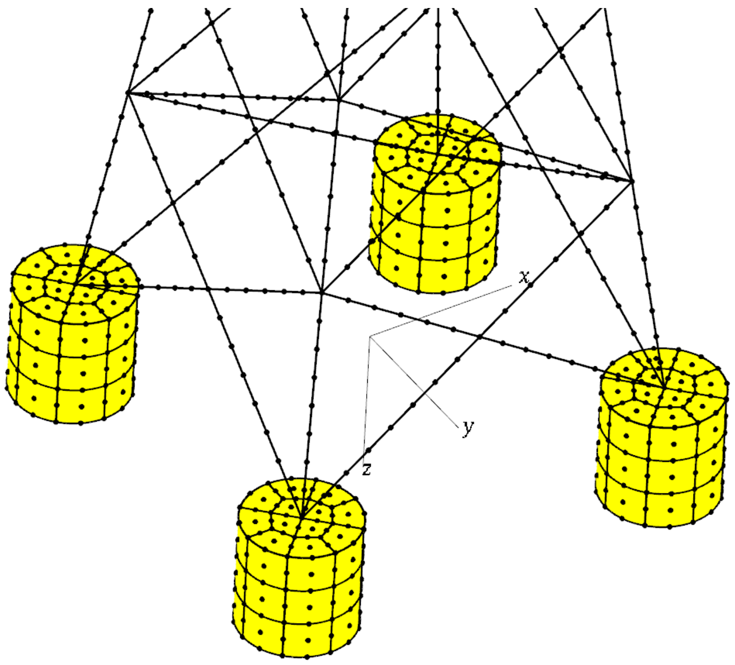


Figure 1.1: Jacket-based Offshore Wind Turbine founded on suction caissons

1.2 How to download

The **binaries** and the **source code** of MultiFEBE are available at the research group webpage and at GitHub. MultiFEBE is released under GPLv2 license. That means that you may copy, distribute and modify the software as long as you track changes/dates in source files. Also, any modifications to or software including (via compiler) GPL-licensed code must also be made available under the GPL along with build & install instructions.

MultiFEBE is available for GNU/Linux and Windows 64-bit operating systems. Binaries are released in `*.deb` and `*.sh` installer packages for GNU/Linux, and in an *.exe installer for Windows. Source code can be compiled using `cmake` + `make` + `gfortran` in GNU/Linux, and through `MSYS2/mingw-w64` + `cmake` + `make` + `gcc-fortran`. The program requires multi-core linear algebra libraries: ATLAS or OpenBLAS; which are successors of single-core LAPACK and BLAS. OpenBLAS is the default choice as it can easily be used in both GNU/Linux and Windows. The whole package is configured as a static build, i.e. the program is statically linked against libraries. Details about installing are given next. The compilation procedure is given in Appendix A.

1.3 How to install

1.3.1 GNU/Linux `*.deb` installer

The installation steps are:

1. Go to GitHub, and download the most recent `*.deb` release.
2. Make double-click on the installer.
3. Press the install button, and then type your password in the authentication window.

1.3.2 GNU/Linux `*.sh` installer

The installation steps are:

1. Go to GitHub, and download the most recent `*.sh` release (e.g. `multifebe-x.x.x-Linux.sh`).
2. Press Ctrl+Alt+T to open a terminal, and navigate to the folder where it has been downloaded.
3. Change the permissions of the file to allow its execution:

   ```
   $ chmod +x multifebe-x.x.x-Linux.sh
   ```

4. Now, you can execute the installer:

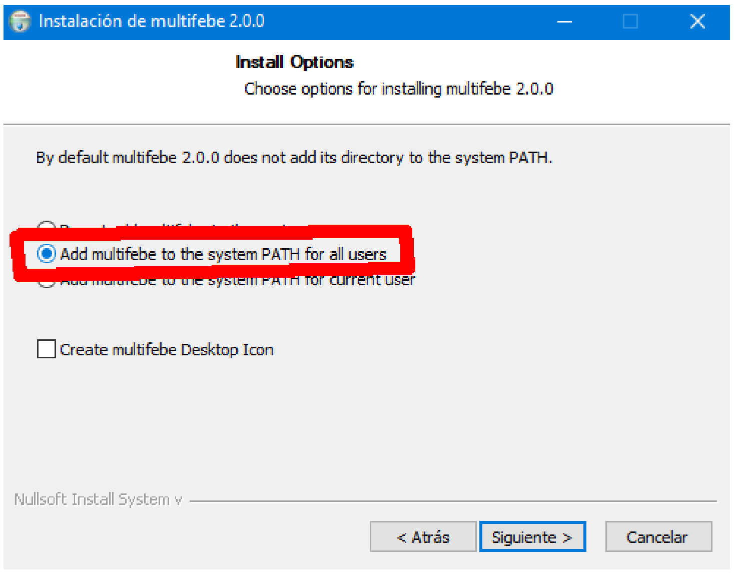
   ```
   $ ./multifebe-x.x.x-Linux.sh
   ```

5. Follow the instructions of the installer. The program files are decompressed by default in the same directory as the installer. You will have to copy or move the binary `multifebe` to `/usr/bin`, or create a simbolic link, so that it is available in the terminal. See more details in Appendix A.

1.3.3 Windows installer

The installation steps are:

1. Go to GitHub, and download the most recent Windows installer release `*.exe`.
2. Right-click on the installer, and **run the application as administrator**.
3. Follow the installer instructions, except in the step shown in Figure 1.2, where you have to add the directory to the system PATH in order to make the executable visible in the terminal.


Instalación de multifebe 2.0.0
Install Options
Choose options for installing multifebe 2.0.0
Nullsoft Install System v
< Atrás
Siguiente >
Cancelar

By default multifebe 2.0.0 does not add its directory to the system PATH.

Do not add multifebe to the system PATH
[✓ Add multifebe to the system PATH for all users]
Add multifebe to the system PATH for current user

Create multifebe Desktop Icon

Figure 1.2: Windows installer: Step 3

4. Continue the installer steps up until the end.

1.4 How to use

MultiFEBE is a command-line program that you can run on a Windows terminal (cmd or PowerShell) or GNU/Linux terminal. Once installed, you can run it by executing `multifebe` from a terminal as we already mentioned at the end of the installation steps. The program execution command line is started by `multifebe` and followed by a set of arguments which tells the program on how to run (options). For instance, as we did before, if we execute:

```
$ multifebe --help
```

it will then print a brief help about how to run it and the available options:

```
$ multifebe --help
Usage:
  multifebe [options]

Options:
  -i, --input STRING        Input file name           [required]
  -o, --output STRING       Output files basename     [default: input file name]
  -m, --memory INTEGER      Max. memory allowed (GB)  [default: 0 (unlimited)]
  -b, --verbose INTEGER     Verbose level: 0-10       [default: 1]
  -v, --version             Program version
  -h, --help                This help
```

Note that each option can start with one or two hyphens, followed by respectively the short or long name of the option. In the previous case, we could request help from the program by writing either `-h` or `--help`. Some options does not require a value like `-h` or `-v`, but other options require a string or a number, which are written after each of them. Options are self-descriptive from the program help, but more details are given in Table 1.1.

Option name			
Short	**Long**	**Argument**	**Description**
`-i`	`--input`	`STRING`	`STRING` is a path to the plain text input file where the definition of the case is read. This option is thus mandatory.
`-o`	`--output`	`STRING`	`STRING` is a path to the basename of the output files (plain text) where the results of the case are printed. This option is optional since the input file path is taken by default. The output files differ only in the extension of the file.
`-m`	`--memory`	`INTEGER`	`INTEGER` is the maximum memory (in GB) allowed to be used by the program when allocating the main matrices. If the program predicts that such memory (or more) is required, then it stops itself. This option is important when dealing with large models in order to prevent the computer from memory swapping (transferring memory data to the hard-drive). Therefore, it is recommended to use this option with a setting of around 75% of the available memory of the computer.
`-b`	`--verbose`	`INTEGER`	`INTEGER` sets the level of details printed by the program when executing, being 0 the level where minimal information is printed and 10 the level where all the steps are printed.

Table 1.1: MultiFEBE command-line arguments (options)

MultiFEBE uses OpenMP in order to take full advantage of multi-core shared-memory computers. By default, the program uses all the available cores, but it is possible to control the number of cores used by previously defining the environment variable `OMP_NUM_THREADS`. Imagine you would like to only use 4 cores, then in GNU/Linux you would have to execute:

```
$ export OMP_NUM_THREADS=4
```

In Windows, it would have to execute:

```
$ set OMP_NUM_THREADS=4
```

Figure 1.3 shows a MultiFEBE usage flowchart. It operates mainly through plain text files. The program reads the input file in order to define the case (type of analysis, geometry topology, mesh, materials, boundary conditions, results to be calculated, etc). Results are written into a set of output files which depends on the type of results requested in the input file.

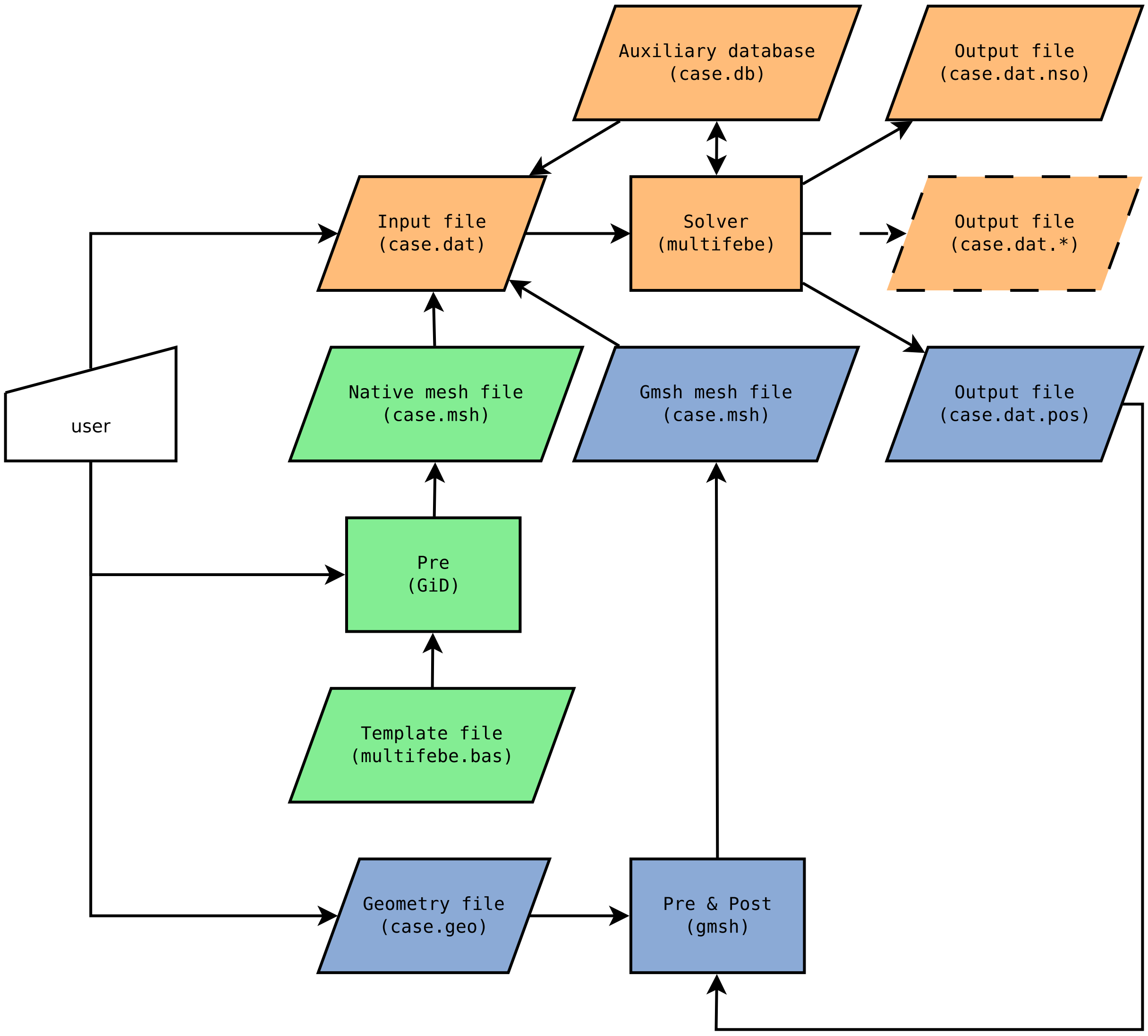

Auxiliary database (case.db)
Output file (case.dat.nso)
Input file (case.dat)
Solver (multifebe)
Output file (case.dat.*)
user
Native mesh file (case.msh)
Gmsh mesh file (case.msh)
Output file (case.dat.pos)
Pre (GiD)
Template file (multifebe.bas)
Geometry file (case.geo)
Pre & Post (gmsh)

Figure 1.3: MultiFEBE usage flowchart. Orange blocks corresponds to MultiFEBE solver files and processes. Blue blocks corresponds to Gmsh pre- and post-processor files and processes. MultiFEBE is able to read Gmsh mesh files and write Gmsh postprocessing results files. Green blocks corresponds to the possibility of generating the mesh in MultiFEBE native format from GiD and a `*.bas` template file.

Most of the user effort is usually spent in preparing the geometry, mesh and then post-processing the results. In order to facilitate these tasks, we have included the possibility of reading and writing Gmsh files. Gmsh is a powerful free and open-source pre- and post-processor. The only Gmsh file format that MultiFEBE can read is MSH file format version 2.2, which is the default file format up to Gmsh 3.0.6. Newer Gmsh versions save mesh files in newer mesh file formats by default. However, you can also export meshes in older file format versions by doing `File>Export>Format: Mesh - Gmsh MSH (*.msh)>Format: Version 2 ASCII` when using the GUI, or including the option `-format msh2` when using the command-line. This integration enables the user to perform, for example, extensive parametric studies with relatively little extra effort, and effective visualization of results. We have also created a GiD `*.bas` template file which allows exporting the mesh file in the MultiFEBE native format (located in the `bin/` folder). We intend to improve the integration of MultiFEBE solver in GiD and others pre- and post-processors in the near future. In the same vein as PILEDYN, we have also created some problem-specific `https://www.mathworks.com/`MATLAB pre- and post-processors that will be available in our research group webpage.

The proposed workflow to perform an analysis is:

1. Decide the modelling approach, which will lead to the required topology and geometry. We suggest using Gmsh for creating the geometry, which allows defining the geometry parametrically through simple `.geo` files.

2. Generate the mesh with appropriate element sizes. In Gmsh, this can be easily done by defining element sizes all over the geometry using variables. Then you can set their value in order to generate a mesh. Once you have your solver results, you may have to return to this step to generate additional refined meshes in order to check results convergence.

3. Open your favourite plain text text editor, and write the case input file according to the required data, as shown in Chapter 2. If you use Gmsh for generating the mesh file, then you will have to define the path to it in section `[settings]`.

4. Run the solver. Copy the solver executable to the folder where the case input file is located. Then, open a terminal in this folder and execute:

   ```
   $ multifebe -i case.dat
   ```

   where it has been assumed that the case input file is named `case.dat`. BEM models may require a huge amount of memory. We recommend that if the model is relatively big, or you are running the solver on a laptop, then you have to limit the available RAM for the program by including the following option:

   ```
   $ multifebe -i case.dat -m 4
   ```

   which will stop the solver if the predicted required memory is greater that 4 GB (or any other quantity smaller than the available memory in the computer). This will prevent the computer from swapping and halting if such amount of memory is required.

5. Once the solver ends, the results are exported to one or more output files, see Chapter 3. If you used a Gmsh mesh file (`*.msh`), Gmsh post-processing files `*.pos` will be generated. If that is the case, you can visualize the results in the Gmsh GUI. Other text files containing node-by-node, element-by-element, or other results can be generated (depending on how you set the case input file). This text files can be easily processed by spreadsheet programs, MATLAB/Octave, GNU utilities such as awk, etcetera. We recommend the latter, as it is particularly fast once you learn how to use it. They can also be plotted through GNU/Linux utilities such as GNUPlot.

Chapter 2

Input file

2.1 Introduction

The input file is a **plain text** file which contains the **case data**. In the following, we will assume that the name of the input file name is `input.dat`, but any other name and extension will also be valid. The important fact is that the file must be a plain text file, and thus it must be edited as such in plain text editors such as Notepad, Notepad++ or others in Windows (but not Word or LibreOffice!), and emacs, gedit, vim or others in GNU/Linux environments.

The case data is divided into **sections**. Each section is a block of data related to an aspect of the case or the solver settings. Each section begins with a line exclusively containing a header with the section name between brackets, i.e. `[section name]`, and it ends when all the required data has been introduced or when another section header appears. It is important to note that its location within the file is not relevant. The data required for a section is defined by one or more lines after the header line. Each section has its own section format, which can be in the form of simple statements (`keyword = values`, `entity id: values`), or a sequence of numbers and strings. Only the recognized sections are read. Therefore, comments can be introduced outside the recognized ones (see example below). Recognized sections are shown in Table 2.1. Some sections are required (mandatory), some sections are required depending on the data introduced in other sections (dependant), and others are completely optional.

input.dat

```
[case description] <-- this is an unrecognized section used as a comment section
3x3 pile group analysis of stresses

[problem]
n = 3D
type = mechanics
analysis = harmonic

[frequencies]
Hz
lin
100
0.01
10.

[symmetry planes]
plane_zx: symmetry
plane_yz: antisymmetry

...
```

All the identifiers of model entities (regions, boundaries, elements, nodes, etc) must be integers greater than 0. Floating point numbers can be expressed as `125.` (simple precision), `1.25e2` (simple precision) or `1.25d2` (double precision). Complex numbers are expressed as `(2.d3,1.d1)`.

Section name	**Brief description**	**Character**
	General sections	
`problem`	Type of problem, dimension $\mathbb{R}^n$ of the ambient space, and analysis to be performed	mandatory
`frequencies`	Frequencies for a time harmonic analysis	dependent
`settings`	Solver settings	optional
`export`	Export options and settings	optional
`materials`	Materials to be used in the model	optional
	Low-level entities of the model (mesh) in native format	
`nodes`	Nodes of the mesh	dependant
`elements`	Elements of the mesh	dependant
`parts`	Parts of the mesh (a connected set of elements of the same intrinsic dimension)	dependant
	High-level entities of the model (topology)	
`boundaries`	Boundaries of BE regions (a part with elements of intrinsic dimension $\mathbb{R}^{n-1}$)	dependant
`be body loads`	Body loads of BE regions (a part with elements of intrinsic dimension from 0D (point) to $\mathbb{R}^n$)	dependant
`fe subregions`	Subregions of FE regions (a part with elements of intrinsic dimension $\mathbb{R}^n$ (solid) or lower (structure))	dependant
`cross sections`	Cross sections of the model. It contains data which complements the geometrical information contained in the mesh. It is required for structural elements.	dependant
`regions`	Regions (or subdomains) of the model	mandatory
	Auxiliary entities of the model	
`internal points`	Internal points of BE regions where the solution is calculated	optional
`groups`	Group of entities of the same class	optional
	Conditions throughout the model	
`symmetry planes`	Symmetry planes of the model	optional
`conditions over be boundaries`	Boundary conditions of BE boundaries	dependent
`conditions over fe elements`	Boundary conditions of FE elements	optional
`conditions over nodes`	Boundary conditions of BE or FE nodes	optional
`incident waves`	Input or incident wave fields	dependant

Table 2.1: Sections of the data file

2.2 General sections

2.2.1 Section `problem`

This section defines the main aspects of the case. It is composed of one or more unordered lines containing one assignment `keyword = values`. Table 2.2 shows the recognized keywords, the values they can take, and a brief description of them.

Keyword	**Requirement**	**Values**	**Description**
`n`	mandatory	`2` or `2D`	Two-dimensional ambient space ($\mathbb{R}^2$)
		`3` or `3D`	Three-dimensional ambient space ($\mathbb{R}^3$)
`type`	mandatory	`laplace`	Laplace problem (unmaintained, it may not work properly)
		`mechanics`	Mechanics problem
`subtype`	dependant	`plane_strain`	Plane strain problem (only for mechanics in $\mathbb{R}^2$)
		`plane_stress`	Plane stress problem (only for mechanics in $\mathbb{R}^2$)
`analysis`	mandatory	`static`	Static analysis
		`harmonic`	Time harmonic analysis

Table 2.2: List of recognized keywords and values in section `problem`

In the following example, a static analysis of a plane strain problem is established:

input.dat
```
...

[problem]
n = 2D
type = mechanics
subtype = plane_strain
analysis = static

...
```

2.2.2 Section `frequencies`

This section is required when `analysis = harmonic` in section `[problem]`. That is because, for a time harmonic analysis, it is necessary to specify the frequencies to be analyzed, which are defined in this section. This section can be present in the input file, or in another auxiliary file whose path is defined in section `[settings]` (see section 2.2.3).

The section format is a sequence of numbers and strings. The first line is a string which indicates the units of the frequency, and it must be `Hz` (cycles per second) or `rad/s` (radians per second). The second line indicates how frequencies are defined, and it must be one of the following:

- `list` - Frequencies are defined by a given list of frequencies. The list can be unordered.
- `lin` - Frequencies are defined by a given number of uniformly (linearly) distributed frequencies between a minimum and a maximum.
- `log` - Frequencies are defined by a given number of logarithmically distributed frequencies between a minimum and a maximum.

The third line defines the number of frequencies. If the frequencies are defined by a `list`, these are defined in the fourth and the following lines (as much as the number of frequencies). If the frequencies are defined by a `lin` or `log` distribution of frequencies, the fourth and fifth lines respectively contains the minimum and maximum frequency. The general format of the section depending on how frequencies are defined (`list`, `lin` or `log`) can be illustrated as:

```
input.dat
...

[frequencies]
<units = Hz or rad/s>
list
<n = number of frequencies>
<frequency 1>
<frequency 2>
...
<frequency n>
...
```

```
input.dat
...

[frequencies]
<units = Hz or rad/s>
lin
<n = number of frequencies>
<minimum frequency>
<maximum frequency>


...
```

```
input.dat
...

[frequencies]
<units = Hz or rad/s>
log
<n = number of frequencies>
<minimum frequency>
<maximum frequency>


...
```

In the following, we are going the show three example of each way of defining the frequencies. In the first place, we show how to define an analysis at frequencies 9 Hz, 2 Hz, 5 Hz and 4 Hz. Secondly, we show how to define an analysis at 10 frequencies linearly distributed between 10 and 100 rad/s, i.e. 10 rad/s, 20 rad/s, ..., 100 rad/s. Last, we show how to define an analysis at 4 frequencies logarithmically distributed between 10 and 10000 Hz, i.e. 10 Hz, 100 Hz, 1000 Hz and 10000 Hz.

```
input.dat (example of list)
...

[frequencies]
Hz
list
4
9.
2.
5.
4.

...
```

```
input.dat (example of lin)
...

[frequencies]
rad/s
lin
10
10.
100.




...
```

```
input.dat (example of log)
...

[frequencies]
Hz
log
4
10.
10000.




...
```

2.2.3 Section `settings`

This section defines several settings of the solver. It is optional since all the parameters here defined have reasonable default values. It is composed of one or more lines containing an assignment `keyword = values`. Table 2.3 shows the recognized settings and a brief description of each one. These are related to the boundary element integration, geometrical parameters, solving of linear system of equations and auxiliary files definition.

Boundary element integration settings have an impact on accuracy and computational cost. It is not recommended to tweak these values unless you have certain specific needs. For example, if you would like to obtain very accurate results, then you can use `qsi_relative_error=1d-12`, so all quasi-singular (or nearly-singular) integrals are calculated with such accuracy. This obviously has an important impact on the computational costs related to building the linear system of equations. Singular integrals are evaluated with high accuracy by default (10^{-12}), and it cannot be adjusted by the user. On the other hand, if you would like to obtain just some fast results, you can use `qsi_relative_error=1d-3`. Note that boundary element integration accuracy is not the only source of errors, so do not expect to have such relative errors in the final results. The parameter `qsi_ns_max` defines the maximum number of subdivisions allowed when performing quasi-singular integration, but note that the quasi-singular integration strategy combines Telles's transformation and subdivision, so the default value of 16 should never be reach unless some very picky calculation is requested. The last setting `precalsets` has impact only on performance, since what it defines is the integration rules where some components of the integrand are precalculated and saved in memory.

The geometrical settings are two: the geometrical tolerance for detecting contact between geometrical entities, and and option which internally collapse the position of nodes sharing the same position within the geometrical tolerance. You should tweak these parameters when the size of the problem is very small or very large, since the geometrical tolerance is defined in absolute and not relative terms with respect to the element or mesh size.

The settings related to the linear system of equations solving allows tweaking how they are solved. You should use scaling and refining when the condition number is very large, which may happen for coupled boundary element - finite element models where regions with very different stiffnesses are connected. This obviously has some impact on the computational cost, but it is a price to pay in certain cases.

The settings related to auxiliary files allows reading some file sections from other files. In the first place, you can define the path to a **file containing the mesh**. This file can be in two different file

Keyword	**Values**	**Default**	**Description**
		Boundary element integration	
`qsi_relative_error`	$\in [10^{-15}, 10^{-3}]$	10^{-6}	Relative error when evaluating quasi-singular integrals (integration of boundary elements).
`qsi_ns_max`	≥ 0	16	Maximum number of subdivisions of the integration domain when evaluating quasi-singular integrals (integration of boundary elements).
`precalsets`	`n d_1 ... d_n`	`8 2 3 4 5 6 7 8 9`	Define the number `n` of pre-calculated datasets and the number `d_j` of 1D Gauss-Legendre quadrature points to be used for each dataset `j` (pre-calculation of integrand components of boundary elements).
		Geometrical parameters	
`geometric_tolerance`	> 0	10^{-6}	Geometric tolerance for detecting contact between geometrical entities. It has the same units as the mesh coordinates, i.e. no relative geometrical tolerance with respect to element/mesh size is performed.
`collapse_nodal_pos`	`T` or `F`	`T`	For each node, a ball of radius equal to the geometric tolerance and centered at the node is built, and all nodes inside this ball are moved to their center of mass
		Linear system of equations	
`lse_straight`	`T` or `F`	`T`	Perform a direct solving of the system of linear equations
`lse_scaling`	`T` or `F`	`F`	Perform scaling when solving of the system of linear equations
`lse_condition`	`T` or `F`	`F`	Estimate the condition number when solving of the system of linear equations
`lse_refine`	`T` or `F`	`F`	Refine the solution after solving of the system of linear equations
		Auxiliary files	
`mesh_file_mode`	`mode file`	`0`	Establish how the mesh is read. If `mode=0`, then the mesh is read from the input file. If `mode=1` or `mode=2`, then the mesh is read from another file specified by `file`. If `mode=1`, it is assumed that the mesh is in the native format described in the present document. If `mode=2`, then it is assumed that the mesh is in the Gmsh MSH file format version 2.2.
`frequencies_file`	`file`	“”	If defined, frequencies are read from the specified file.‘

Table 2.3: List of recognized settings and values in section `settings`

formats: native or Gmsh (MSH version 2.2). The native format is a very easy to manually generate, but it is also the file format which GiD generates after using the template file `multifebe.bas` (see Appendix B). The Gmsh mesh file format 2.2 is the default file format generated by Gmsh from Gmsh version 2.0.0 up until 3.0.6. Newer versions (Gmsh 4.0.0 onwards) generate by default a new file format version 4. However, you can still export meshes in this older file format versions by doing `File>Export>Format: Mesh - Gmsh MSH (*.msh)>Format: Version 2 ASCII` when using the GUI, or including the option `-format msh2` when using the command-line. You could also read from another file the frequencies, for example, if you save some standard set of frequencies in a file you can use in several cases, e.g. one-third octave middle frequencies for sound analysis.

Most of the times, you would only need to define the path to a mesh file and its format. For example, a Gmsh mesh file located in the same directory as the input file `input.dat`:

input.dat
```
...

[settings]
mesh_file_mode = 2 "pilegroup.msh"

...
```

2.2.4 Section `export`

This section allows the definition of export options. It is optional since all the parameters here defined have reasonable default values. It is composed of one or more lines containing one assignment `keyword = values`. It is meant to configure which and how output files have to be exported. Table 2.4 shows the recognized export options and a brief description of each of them.

User can activate/deactivate exporting general output files such as node-by-node solutions and element-by-element solutions in a native format, a file containing the wave propagation speeds of each boundary element region (particularly useful when using a Biot's poroelastic region), and a Gmsh post-processing file containing mainly all the results. More details about the output file formats are given in Chapter 3.

The user can request the calculation of kinematic and stress resultants over BE boundaries and BE body loads. That means that average kinematics (displacements and rotations) and stress resultants are calculated for each BE boundary and each BE body load present in the model. This is useful, for example, for obtaining soil reactions in soil-structure interaction problems. Note that there are to additional options to this calculation, which are defining the point where the resultant rotations and moments are calculated and if the symmetry configuration is taken into account or not.

Finally, since each file contains basically integers, real and complex numbers, and the output files are plain text files, the user can establish how these are written in the file. Files really only contain integers and reals, whose number of digits, notation, etc, can be defined via keywords `integer_format` and `real_format` respectively. Complex numbers are written as two consecutive real numbers containing whether the absolute value and argument pair (polar notation), or the real part and the imaginary part (cartesian notation).

Most of the times, you would not require to use this section. Perhaps, if you would like to directly have in your output files the complex results in polar notation, and you only require 8 digits after the decimal point (which reduce the file size), you could write the section as:

input.dat
```
...

[settings]
real_format = eng_simple
complex_notation = polar

...
```

2.2.5 Materials

This section allows the definition of a set of materials and its properties. The type of materials available are three linear elastic materials: inviscid fluid, elastic solid, and Biot's poroelastic medium [7]. Each one of these represent a very different kind of material. The first one represents an acoustic medium, where only longitudinal P waves can propagate. It is therefore appropriate to model sound propagation through fluids like air or water. The second one is the common material for modelling structures, soils, etc. In a continuum region, longitudinal P and transverse S waves can propagate. The last one represent

Keyword	Values	Default	Description
		General output files	
`export_wsp`	`T` or `F`	`F`	Export wave propagation speeds in each boundary element region.
`export_nso`	`T` or `F`	`T`	Export nodal solutions
`export_eso`	`T` or `F`	`T`	Export element solutions
`export_pos`	`T` or `F`	-	Export results in Gmsh MSH file format version 2.2. The default behaviour is `F`, but it is automatically set to `T` if `mesh_file_mode=2` (i.e. a Gmsh mesh file is read).
		BE boundaries and BE body loads kinematic and stress resultants output file	
`export_tot`	`T` or `F`	`F`	Export the resultant average displacements and rotations, and total forces and moments of each BE boundary.
`tot_xm`	`0` or `1`	`0`	If `tot_xm=0`, then the rotations and moments are calculated with respect to the origin ($\mathbf{x}_m = \mathbf{0}$). If `tot_xm=1`, then the moments are calculated with respect to the centroid of each BE boundary.
`tot_apply_symmetry`	`T` or `F`	`T`	If set, kinematic and stress resultants are calculated taking into account the symmetry configuration.
		Export settings for writing integer, real and complex numbers.	
`real_format`	`f<W>.<d>`	`eng_double`	A Fortran-like string (descriptor) with the format for floating point numbers must be used, see Table 2.5.
	`e<W>.<d>e<E>`		
	`en<W>.<d>e<E>`		
	`sci_double` (`e25.16e3`)		
	`sci_simple` (`e16.8e2`)		
	`sci_less` (`e11.3e2`)		
	`eng_double` (`en27.16e3`)		
	`eng_simple` (`en18.8e2`)		
	`eng_less` (`en13.3e2`)		
`integer_format`	`I<W>`	`auto`	A Fortran-like string (descriptor) with the format for integer numbers must be used, see Table 2.5.
	`max` (`i11`)		
	`auto` (`<W>` = max id length)		
`complex_notation`	`polar`	`cartesian`	Complex notation to be used when exporting: polar (absolute value and argument), or cartesian (real part and imaginary part).
	`cartesian`		

Table 2.4: List of export options and values in section `export`

Variable		Format descriptor	
Integer		`Iw`	`Iw.m`
Real	Decimal	`Fw.d`	
	Exponential	`Ew.d`	`Ew.dEe`
	Scientific	`ESw.d`	`ESw.dEe`
	Engineering	`ENw.d`	`ENw.dEe`

`w`: the number of positions to be used

`m`: the minimum number of positions to be used

`d`: the number of digits to the right of the decimal point

`e`: the number of digits in the exponent part

Table 2.5: Fortran descriptors

a porous medium where a compressible fluid is present inside a elastic frame. In a continuum region, two types of longitudinal waves (P1 and P2) and transverse S waves can propagate.

The section format is a sequence of numbers and strings, whose format can be described as follows:

general format of section [materials]

```
[materials]
<n = number of materials>
<material 1 identifier> <type of material> <property> <value> <property> <value> ...
<material 2 identifier> <type of material> <property> <value> <property> <value> ...
...
<material n identifier> <type of material> <property> <value> <property> <value> ...
```

The first line must contain the number of materials to be considered. Next, there must be as many lines as the number of materials defined. Each line starts with the material identifier (a integer greater than 0). Then, it follows with a string indicating the type of material to be defined in that line: `fluid`, `elastic_solid`, or `biot_poroelastic_medium`. After defining the type of material, if follows several pairs of string (property symbol) and real value (property value), which depends on the type of material.

If the type of material is `fluid` (which can be used only for time harmonic analysis), then it is necessary to define two between the three following properties: bulk modulus K, density ρ and wave propagation speed c. Optionally, you can define an articial hysteretic damping ratio ξ, which is zero by default. See Table 2.6 for more details.

Property	**Math symbol**	**Plain text symbol**
Bulk modulus	K	`K`
Density	ρ	`rho`
Wave propagation speed	$c = \sqrt{K/\rho}$	`c`
Artificial hysteretic damping ratio	ξ	`xi`

Table 2.6: Properties of an inviscid fluid (`fluid`). Only usable for time harmonic analysis. At least two between K, ρ and c must be defined. ξ is optional ($\xi = 0$ by default).

If the type of material is `elastic_solid`, then it is necessary to define two of the five following properties: Young's modulus E, bulk modulus K, Lamé's first parameter, λ, shear modulus μ and Poisson's ratio ν. If a time harmonic analysis is going to be performed, it is mandatory to define the density. Optionally, you can define the hysteretic damping ratio ξ, which is zero by default. See Table 2.7 for more details.

Property	**Math symbol**	**Plain text symbol**
Young's modulus (elastic modulus)	E	`E`
Bulk modulus	K	`K`
Lamé's first parameter	λ	`lambda`
Lamé's second parameter (shear modulus)	μ	`mu`
Poisson's ratio	ν	`nu`
Density	ρ	`rho`
Hysteretic damping ratio	ξ	`xi`

Table 2.7: Properties of an elastic solid (`elastic_solid`). At least two between E, K, λ, μ and ν must be defined. ρ is mandatory for time harmonic analysis. ξ is optional ($\xi = 0$ by default).

If the type of material is `biot_poroelastic_medium`, then it is necessary to define two of the five following properties of the solid phase: Young's modulus E, bulk modulus K, Lamé's first parameter, λ, shear modulus μ and Poisson's ratio ν. It is mandatory to define the solid and fluid phases densities, as well as the porosity, coupling parameters, additional density and dissipation constant. Optionally, you can define the solid phase hysteretic damping ratio ξ, which is zero by default. See Table 2.8 for more details.

Let's show a very simple example where we have two materials: water (c = 1480 m/s, ρ = 1000 kg/m^3) and soil modelled as an elastic solid (E = 60 MPa, ν = 0.4, ρ = 2000 kg/m^3, ξ = 5%). In such a case, the section should be written as:

Property	**Symbol**	**Plain text symbol**
Solid phase		
Young's modulus (elastic modulus)	E	`E`
Bulk modulus	K	`K`
Lamé's first parameter	λ	`lambda`
Lamé's second parameter (shear modulus)	μ	`mu`
Poisson's ratio	ν	`nu`
Hysteretic damping ratio	ξ	`xi`
Solid density	ρ_s	`rhos`
Fluid phase and coupling parameters		
Fluid density	ρ_f	`rhof`
Porosity	ϕ	`phi`
Biot's coupling parameter	Q	`Q`
Biot's coupling parameter	R	`R`
Additional apparent density	ρ_a	`rhoa`
Dissipation constant	b	`b`

Table 2.8: Properties of a Biot's poroelastic medium (`biot_poroelastic_medium`). At least two between E, K, λ, μ and ν must be defined. ξ is optional ($\xi = 0$ by default). The other properties are mandatory.

`input.dat`

```
...

[materials]
2
1 fluid c 1480. rho 1000.
2 elastic_solid E 60.d6 nu 0.4 rho 2000. xi 0.05

...
```

2.3 Low-level entities of the model (mesh) in native format

The low-level entities of the model are the elementary entities (mesh) of the model: nodes, elements and parts. By default, the mesh is read from the input file by writing by hand the sections `[nodes]`, `[elements]` and `[parts]`, as explained below. However, there are other ways to create and read the mesh.

We have written a template file `*.bas` for the GiD pre- and post-processor, which allows GiD to produce a mesh file in our native mesh file format. Then, you can copy the contents of the generated file to the input file, or use `mesh_file_mode = 1 "filepath"` option in `[settings]` to indicate the format and path to the file. Each "layer" in GiD jargon corresponds to our concept of "part".

The program can read directly a mesh file from the Gmsh pre- and post-processor (MSH file format version 2.2), by using the `mesh_file_mode = 2 "filepath"` option in `[settings]` to indicate the format and path to the file. Each "physical" entity in their Gmsh jargon corresponds to our concept of "part".

2.3.1 Section `nodes`

In this section, all the nodes of the model are defined. Nodes are the most elementary part of the mesh, and they carry geometrical information (position of nodes), but also a functional/physical information (value of displacement, traction, etc. at that position).

The first line indicates the number of nodes. Then, one line per node indicating the identifier of the node and its coordinates. The general format of this section is:

general format of section [nodes]
```
[nodes]
<n = number of nodes>
<node 1 identifier> <x> <y> <z>
<node 2 identifier> <x> <y> <z>
...
<node n identifier> <x> <y> <z>
```

2.3.2 Section `elements`

In this section, all the elements of the model are defined. The elements are the fundamental part of the mesh, they allow the definition of an interpolation of the geometry and physical variables (displacements, tractions, etc.) supported on the nodes. On the other hand, high-level entities like boundaries and subdomains are built using a connected set of elements, which here it is called a "part".

The format of this section is very similar to the corresponding section of the Gmsh file format. The first line indicates the number of elements. Then, one line per element indicating:

- Element identifier.
- Type of element. It could be introduced via a string: `line2`, `line3`, `tri3`, `tri6`, `quad4`, `quad8`, `quad9`; or a number: `1`, `8`, `2`, `9`, `3`, `16`, `10`; respectively. Note that the numbers correspond to the numbers used by the `gmsh` file format.
- Number of auxiliary tags (greater than 0).
- List of tags, where the first auxiliary tag is mandatory, and corresponds to the identifier of the part which the element belongs. The rest of the tags are optional and they are read, but they are not used.
- A list of identifiers corresponding to the nodes of the element.

The general format of this section is:

general format of section [elements]
```
[elements]
<n = number of elements>
<element 1 identifier> <type> <num. of tags> <list of tags> <list of nodes identifiers>
<element 2 identifier> <type> <num. of tags> <list of tags> <list of nodes identifiers>
...
<element n identifier> <type> <num. of tags> <list of tags> <list of nodes identifiers>
```

Let's show an example of a mesh with 10 elements of the type `line3` (quadratic line element), where elements 1 to 5 belongs to part 1, and elements 6 to 10 belongs to part 2:

input.dat
```
...

[elements]
10
 3 line3 1 1  1  3  2
 2 line3 1 1  3  5  4
 1 line3 1 1  5  7  6
 4 line3 1 1  7  9  8
 5 line3 1 1  9 11 10
 6 line3 1 2 12 14 13
10 line3 1 2 14 16 15
 8 line3 1 2 16 18 17
 9 line3 1 2 18 20 19
 7 line3 1 2 20 22 21

...
```

Note that given that we use an identifier for each element, the order is unimportant. The same happens with other model entities such as nodes and parts.

2.3.3 Section parts

In this section, all the parts of the model are defined. A part is a connected set of elements. Hence, an element belongs only to one part, and one part can contain several elements. Note that the relationship between elements and parts was done when defining the elements. The general format of this section is:

```
general format of section [parts]
[parts]
<n = number of parts>
<part 1 identifier> <part name>
<part 2 identifier> <part name>
...
<part n identifier> <part name>
```

where `<part name>` is a string which allows labelling the part for easy identification. However, this is not used in the rest of the case input file.

2.4 High-level entities of the model (topology)

The high-level entities of the model are the classical topological entities: boundaries, and regions (or subdomains); which are used to assign material properties and conditions to different parts of the model. For convenience, four entities are defined: `boundaries` (boundary element boundaries), `be bodyloads` (body loads within a boundary element region), `fe subregions` (finite element subregions) and `regions`. These are defined by writing the corresponding sections `[boundaries]`, `[be bodyloads]`, `[fe subregions]` and `[regions]`, which are explained in the below.

2.4.1 Section boundaries

In a n-dimensional problem, a boundary Γ is a $(n-1)$-dimensional oriented entity that is a part or the whole boundary of a region. The whole boundary $\partial\Omega$ of a region Ω treated by the Boundary Element Method, BE region in the following, is defined as a set of boundaries:

$$\partial\Omega = \{\dots,\Gamma_j,\dots\} \tag{2.1}$$

whose orientation must be outwards from the BE region. A minus sign before Γ_j can be used to indicate the reversion of the orientation of Γ_j in order to get a compatible $\partial\Omega$.

There are two main classes of boundaries:

- **Ordinary.** An ordinary boundary is a boundary in the classical sense. It can be connected with one or two BE regions. In both cases, the boundary can be connected with FE elements.

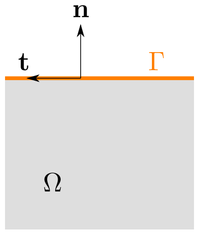


Figure 2.1: Ordinary boundary connected with one region

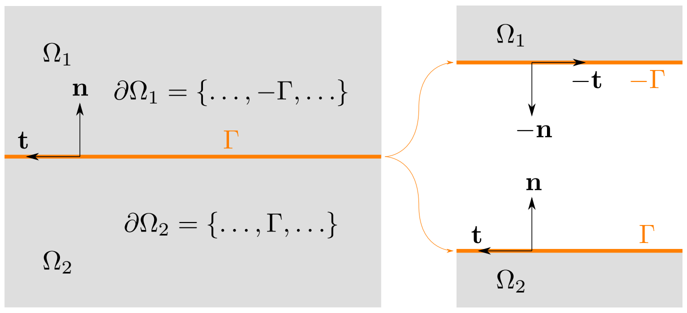
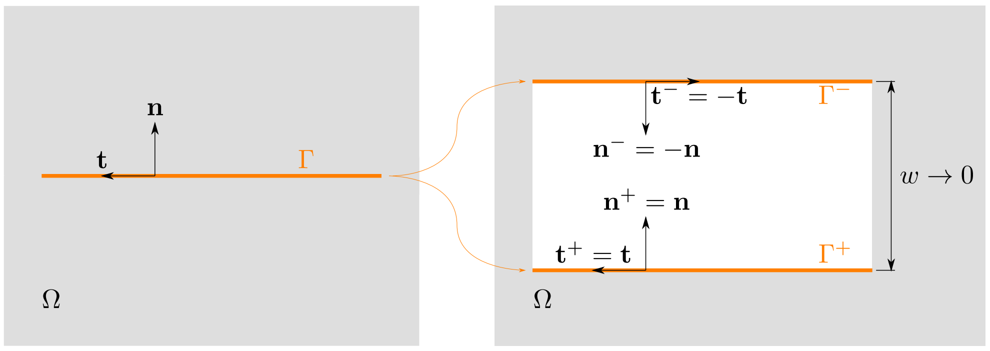


Figure 2.2: Ordinary boundary connected with two regions

- **Crack-like.** A special boundary that lies inside a BE region and is composed by two crack-like sub-boundaries (Γ^+ and Γ^-) of opposite orientations. Thus, a crack-like boundary can be connected only with one BE region. Generally speaking, it is the condensed geometric description of a null thickness inclusion or void. Its orientation defines the orientation of the positive sub-boundary Γ^+, i.e. it defines which face is Γ^+ and which face is Γ^-.



$\mathbf{n}$
$\mathbf{t}$
Γ
Ω
$\mathbf{t}^- = -\mathbf{t}$
Γ^-
$\mathbf{n}^- = -\mathbf{n}$
$\mathbf{n}^+ = \mathbf{n}$
$\mathbf{t}^+ = \mathbf{t}$
Γ^+
$w \to 0$
Ω

Figure 2.3: Crack-like boundary

Each boundary is build up with unique boundary elements and nodes, which are defined by a “part” of the mesh. Thus, all boundaries must be connected with one and only one mesh part. The orientation of all boundary elements in that part must be the same, i.e. be compatible, which defines the orientation of the boundary.

The first line indicates the number of boundaries. Then, one line per boundary indicating the boundary identifier, the identifier of the part that discretize it, and finally the boundary class (`ordinary` or `crack-like`). The general format of the section is:

`general format of section [boundaries]`
```
[boundaries]
<n = number of boundaries>
<boundary 1 identifier> <part identifier> <boundary class: ordinary or crack-like>
<boundary 2 identifier> <part identifier> <boundary class: ordinary or crack-like>
...
<boundary n identifier> <part identifier> <boundary class: ordinary or crack-like>
```

Next, it is shown an example with 4 boundaries, where boundary 1 is the part 1 of the mesh and is an ordinary boundary, boundary 3 is the part 6 of the mesh and is an ordinary boundary, boundary 4 is the part 2 of the mesh and is a crack-like boundary, and boundary 2 is the part 3 of the mesh and is an ordinary boundary:

`input.dat`
```
...

[boundaries]
4
1 1 ordinary
3 6 ordinary
4 2 crack-like
2 3 ordinary

...
```

2.4.2 Section `be bodyloads`

Body loads in BE regions are defined in this section. The general format follows a simpler pattern as the previous section. The first line contains the number of BE body loads to be defined, next as many lines are BE body loads. Each line contains first the BE body load identifier, and last the mesh part which contains the elements associated with it. The general format can be written as:

`general format of section [fe subregions]`
```
[be bodyloads]
<n = number of BE body loads>
<BE bodyload 1 identifier> <part identifier>
<BE bodyload 2 identifier> <part identifier>
...
<BE bodyload n identifier> <part identifier>
```

where the last two zeros at the end of the each line are mandatory, and they are going to be used in the future for additional features.

2.4.3 Section `fe subregions`

A FE subregion is a partition of a FE region. A FE subregion is associated with a part of the mesh. The general format of the section is:

general format of section [fe subregions]
```
[fe subregions]
<n = number of FE subregions>
<FE subregion 1 identifier> <part identifier> 0 0
<FE subregion 2 identifier> <part identifier> 0 0
...
<FE subregion n identifier> <part identifier> 0 0
```

where the last two zeros at the end of the each line are mandatory, and they are going to be used in the future for additional features.

2.4.4 Section `regions`

In a n-dimensional problem, a region (or subdomain) Ω is a n-dimensional entity that is a part or the whole domain of the problem. In MultiFEBE, a region Ω can be discretized by one of two different methods:

- **BEM (BE region).** The region is treated by the BEM. Its discretization is defined by its boundary $\partial\Omega$, which in general is built using a set of boundaries, which in turn is associated with mesh parts and hence with boundary elements.
- **FEM (FE region).** The region is treated by the FEM. Its discretization is defined by a set of FE subregions, which in turn is associated with parts and hence with finite elements. FE region can only be made of elastic solid materials, i.e. finite elements for fluid or poroelastic medium are not availabe.

The interaction (coupling) between BE regions is done through their shared boundaries, which must have opposite orientation for each region, see Figure 2.2. The interaction between FE regions is done through their shared nodes as usual. The interaction between BE boundaries and FE regions is done through the BE boundaries and the boundary of the FE elements that shares the same position. The interaction between BE and FE elements is automatically detected.

The format of this section is explained in the following. The first line indicates the number of regions. Then, for each region there must be a block of data consisting of several lines of data. The first one is the region identifier and the region class (discretization method: “fe” or “be”). If the region is a BE region, then the second line indicates the number of boundaries and a list of boundaries (with their orientation signs). If the region is a FE reigon, then the second line indicates the number of FE subregions and a list of fe subregions. The third line defines the material. Then, only if the region is a BE region, the fourth line defines the number and a list of BE body loads. Also, only if the region is a BE region and the analysis is time harmonic, the fifth line defines the number and a list of incident fields. The general format of the section is:

general format of section [regions]
```
[regions]
<n = number of regions>

<region 1 identifier> <region class (discretization method): be or fe>
<number of boundaries or fe subregions> <list of boundaries or fe subregions identifiers>
material <list of materials>
[<if be: number of be body loads> <list of be body loads>]
[<if be and analysis=harmonic: number of incident fields> <list of incident fields>]

<region 2 identifier> <region class (discretization method): be or fe>
<number of boundaries or fe subregions> <list of boundaries or fe subregions identifiers>
material <list of materials>
[<if be: number of be body loads> <list of be body loads>]
[<if be and analysis=harmonic: number of incident fields> <list of incident fields>]

...

<region n identifier> <region class (discretization method): be or fe>
<number of boundaries or fe subregions> <list of boundaries or fe subregions identifiers>
material <list of materials>
```

```
[<if be: number of be body loads> <list of be body loads>]
[<if be and analysis=harmonic: number of incident fields> <list of incident fields>]
```

2.5 Cross sections

In this section, the additional data required for structural elements (elements simplifying the physics and the geometrical description of continuum media: beams, discrete springs, point masses, etcetera) is introduced. Both the model/theory used for the structural elements and the additional data is assigned at the same time.

The first line of the section defines the number of cross section / structural elements to be defined. Next, there must be as many lines as the number of cross section / structural elements defined. Each line starts first with the type of cross section / structural element to be defined in this line. Next, the number of FE subregions where this type of cross section is assigned, followed by the list of FE subregions. Next, depending on the type of cross section / structural element, different parameters have to be defined. Therefore, the general format of this section is:

```
general format of section [regions]
[cross sections]
<n = number of cross sections>

<type of cross section 1> <m = n. of FE subregions> <subr. 1 id> <subr. 2 id> ... <subr. m id> ...
<type of cross section 2> <m = n. of FE subregions> <subr. 1 id> <subr. 2 id> ... <subr. m id> ...
...
<type of cross section n> <m = n. of FE subregions> <subr. 1 id> <subr. 2 id> ... <subr. m id> ...
```

2.5.1 Discrete translational springs/dashpots (distra)

This type of finite elements are defined in the case input file as `distra`. Discrete translational springs and dashpots (the latter only appearing in time harmonic analysis in parallel to the spring) are introduced in the mesh via 2-node elements. Each element node have 2 or 3 translational degrees freedom respectively for two- and three-dimensional analyses. Instead of introducing a simple spring/dashpot in the direction of the element, a generalized spring/dashpot relating relative displacements of both nodes is used. It comes in two flavors:

- **Local coordinates**. It allows to establish the stiffnesses and viscous damping coeficients in local coordinates, i.e. axial spring/dashpot in x' direction ($k_{x'}$, $c_{x'}$) and lateral springs in y' and z' directions ($k_{y'}$, $c_{y'}$, $k_{z'}$, $c_{z'}$). In two-dimensional analysis, the local stiffness and damping coefficient matrices are:

$$\mathbf{K}' = \begin{bmatrix} k_{x'} & 0 & -k_{x'} & 0 \\ 0 & k_{y'} & 0 & -k_{y'} \\ -k_{x'} & 0 & k_{x'} & 0 \\ 0 & -k_{y'} & 0 & k_{y'} \end{bmatrix}, \quad \mathbf{C}' = \begin{bmatrix} c_{x'} & 0 & -c_{x'} & 0 \\ 0 & c_{y'} & 0 & -c_{y'} \\ -c_{x'} & 0 & c_{x'} & 0 \\ 0 & -c_{y'} & 0 & c_{y'} \end{bmatrix} \tag{2.2}$$

  In three-dimensional analysis:

$$\mathbf{K}' = \begin{bmatrix} k_{x'} & 0 & 0 & -k_{x'} & 0 & 0 \\ 0 & k_{y'} & 0 & 0 & -k_{y'} & 0 \\ 0 & 0 & k_{z'} & 0 & 0 & -k_{z'} \\ -k_{x'} & 0 & 0 & k_{x'} & 0 & 0 \\ 0 & -k_{y'} & 0 & 0 & k_{y'} & 0 \\ 0 & 0 & -k_{z'} & 0 & 0 & k_{z'} \end{bmatrix}, \quad \mathbf{C}' = \begin{bmatrix} c_{x'} & 0 & 0 & -c_{x'} & 0 & 0 \\ 0 & c_{y'} & 0 & 0 & -c_{y'} & 0 \\ 0 & 0 & c_{z'} & 0 & 0 & -c_{z'} \\ -c_{x'} & 0 & 0 & c_{x'} & 0 & 0 \\ 0 & -c_{y'} & 0 & 0 & c_{y'} & 0 \\ 0 & 0 & -c_{z'} & 0 & 0 & c_{z'} \end{bmatrix} \tag{2.3}$$

  In this latter case, in order to uniquely define the local axis, a reference vector $\mathbf{v}_{2\text{ref}}$ is required for establishing the local y' axis. With all these information, we can define a coordinate transformation matrix to build the definitive global stiffness and viscous damping matrix.

- **Global coordinates**. It allows to establish the stiffnesses and viscous damping coeficients directly in global coordinates, i.e. springs and dashpots in x, y and z directions: k_x, c_x, k_y, c_y, k_z and, c_z. In two-dimensional analysis, the global stiffness and damping coefficient matrices are:

$$\mathbf{K} = \begin{bmatrix} k_x & 0 & -k_x & 0 \\ 0 & k_y & 0 & -k_y \\ -k_x & 0 & k_x & 0 \\ 0 & -k_y & 0 & k_y \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} c_x & 0 & -c_x & 0 \\ 0 & c_y & 0 & -c_y \\ -c_x & 0 & c_x & 0 \\ 0 & -c_y & 0 & c_y \end{bmatrix} \tag{2.4}$$

  In three-dimensional analysis:

$$\mathbf{K} = \begin{bmatrix} k_x & 0 & 0 & -k_x & 0 & 0 \\ 0 & k_y & 0 & 0 & -k_y & 0 \\ 0 & 0 & k_z & 0 & 0 & -k_z \\ -k_x & 0 & 0 & k_x & 0 & 0 \\ 0 & -k_y & 0 & 0 & k_y & 0 \\ 0 & 0 & -k_z & 0 & 0 & k_z \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} c_x & 0 & 0 & -c_x & 0 & 0 \\ 0 & c_y & 0 & 0 & -c_y & 0 \\ 0 & 0 & c_z & 0 & 0 & -c_z \\ -c_x & 0 & 0 & c_x & 0 & 0 \\ 0 & -c_y & 0 & 0 & c_y & 0 \\ 0 & 0 & -c_z & 0 & 0 & c_z \end{bmatrix} \tag{2.5}$$

The syntax for such type of finite element is (`<FE subregions>` includes `<m = n. of FE subregions> <subr. 1 id> <subr. 2 id> ... <subr. m id>`):

- 2D static analysis, distra elements in local coordinates:

```
format of section [cross sections] for distra elements
[cross sections]
...
distra <FE subregions> local <kx'> <ky'>
...
```

- 2D static analysis, distra elements in global coordinates:

```
format of section [cross sections] for distra elements
[cross sections]
...
distra <FE subregions> global <kx> <ky>
...
```

- 3D static analysis, distra elements in local coordinates:

```
format of section [cross sections] for distra elements
[cross sections]
...
distra <FE subregions> local <kx'> <ky'> <kz'> <v2r_x> <v2r_y> <v2r_z>
...
```

- 3D static analysis, distra elements in global coordinates:

```
format of section [cross sections] for distra elements
[cross sections]
...
distra <FE subregions> global <kx> <ky> <kz>
...
```

- 2D time harmonic analysis, distra elements in local coordinates ($k_{x'}$ and $k_{y'}$ as complex numbers):

```
format of section [cross sections] for distra elements
[cross sections]
...
distra <FE subregions> local <kx'> <ky'> <cx'> <cy'>
...
```

- 2D time harmonic analysis, distra elements in global coordinates (k_x and k_y as complex numbers):

format of section [cross sections] for distra elements
```
[cross sections]
...
distra <FE subregions> global <kx> <ky> <cx> <cy>
...
```

- 3D time harmonic analysis, distra elements in local coordinates ($k_{x'}$, $k_{y'}$ and $k_{z'}$ as complex numbers):

format of section [cross sections] for distra elements
```
[cross sections]
...
distra <FE subregions> local <kx'> <ky'> <kz'> <cx'> <cy'> <cz'> <v2r_x> <v2r_y> <v2r_z>
...
```

- 3D time harmonic analysis, distra elements in global coordinates (k_x, k_y and k_z as complex numbers):

format of section [cross sections] for distra elements
```
[cross sections]
...
distra <FE subregions> global <kx> <ky> <kz> <cx> <cy> <cz>
...
```

2.5.2 Discrete rotational/translational springs/dashpots (disrotra)

To be documented.

2.5.3 Point mass (pmass)

To be documented.

2.5.4 Bar finite element (bar)

To be documented.

2.5.5 Beam finite element obtained from the degeneration of the solid (degbeam)

To be documented.

2.5.6 Straight beam finite element (strbeam)

To be documented.

2.5.7 Shell finite element obtained from the degeneration of the solid (degshell)

To be documented.

2.6 Conditions of the model

In this section, the data sections related to the conditions (boundary and interface conditions, loads, symmetry planes) of the model are explained.

This part of the manual is incomplete as it only deals with conditions for elastic solids.

2.6.1 Section `symmetry planes`

If present, in this section the symmetry planes of the problem are defined. All symmetry planes are assumed to be at the origin of coordinates. By using symmetry planes, only a half, quarter or octant of the model has to be discretized. For BE regions, no additional boundary conditions are required. For FE regions, however, this must be done manually over nodes belonging to the symmetry planes in the usual way. In future releases, this will be done automatically.

In order to indicate the existence and the nature of the symmetry, it is necessary to define up to three of the symmetry planes implemented: plane with unit normal $\mathbf{n} = \mathbf{e}_1 = (1, 0, 0)$ (plane yz), plane with unit normal $\mathbf{n} = \mathbf{e}_2 = (0, 1, 0)$ (plane zx), or plane with unit normal $\mathbf{n} = \mathbf{e}_3 = (0, 0, 1)$ (plane xy); where each of one can be either of symmetry or antisymmetry. The general format for the section is:

general format of section [symmetry planes]
```
[symmetry planes]
<"plane_n1", "plane_n2" or "plane_n3"> : <"symmetry" or "antisymmetry">
...
```

For example, if the plane with normal $\mathbf{n} = \mathbf{e}_1 = (1, 0, 0)$ (plane yz) is a symmetry plane, i.e. fields at (x_1, x_2, x_3) are symmetric to fields at $(-x_1, x_2, x_3)$, then:

input.dat
```
...

[symmetry planes]
plane_n1: symmetry

...
```

2.6.2 Section `conditions over be boundaries`

In this section, the boundary conditions applied on the boundary elements boundaries are defined. All boundaries except the interfaces and the boundaries that are coupled with finite elements need to specify their boundary conditions. If not defined, a traction-free boundary is assumed by default. Here, boundary conditions for viscoelastic solids are explained.

There are available four boundary conditions for **ordinary boundaries**, two on global coordinates, and two in local coordinates, see Table 2.9. The two boundary conditions of each group are known displacement and known traction. For each boundary, the user must specify the boundary condition for each coordinate in local coordinates or in global coordinates, but not mixed.

Axes	Type	Name	Values	Equations
global	0	displacement	U	$u_k = U$
	1	traction	T	$t_k = T$
	4	infinitesimal rotation field	center $\mathbf{c}$, axis $\mathbf{a}$, angle θ	$u_k = \theta\,[\mathbf{a} \times (\mathbf{x} - \mathbf{c})] \cdot \mathbf{e}_k$
	10	normal pressure	P	$t_k = P n_k$
local	2	displacement	U	$\mathbf{u} \cdot \mathbf{l}_k = U$
	3	traction	T	$\mathbf{t} \cdot \mathbf{l}_k = T$

Note: $k = 1, \ldots, n$ (global or local axes). Local axes are $\{\mathbf{l}_1, \mathbf{l}_2, \mathbf{l}_3\} = \{\mathbf{n}, \mathbf{t}_1, \mathbf{t}_2\}$.

Table 2.9: Boundary conditions for ordinary boundaries

It is always preferable using the B.C. expressed in global axes (B.C. 0 or 1) when possible because they do not need additional equations. If the boundary is planar and its normal vector is parallel to any global axis, then the B.C. expressed in local axes are not needed at all.

There are available two boundary conditions for each face of a **crack-like boundary**. One establishes a displacement, while the other establishes a traction, see Table 2.10.

Face	Type	Name	Values	Equations
Γ^+	0	displacement	U^+	$u_k^+ = U^+$
	1	traction	T^+	$t_k^+ = T^+$
Γ^-	0	displacement	U^-	$u_k^- = U^-$
	1	traction	T^-	$t_k^- = T^-$

Note: $k = 1, \ldots, n$ (global axes).

Table 2.10: Boundary conditions for crack-like boundaries

The general format of the section is:

general format of section [symmetry planes]
```
[conditions over be boundaries]
boundary <boundary id>: <type x> <value x>
                        <type y> <value y>
                        <type z> <value z>
                        <type x (if crack-like: face -)> <value x (if crack-like: face -)>
                        <type y (if crack-like: face -)> <value y (if crack-like: face -)>
                        <type z (if crack-like: face -)> <value z (if crack-like: face -)>
...
```

where note that for two-dimensional problems, the z lines must be removed. An example of a two-dimensional problem with two boundaries with identifiers `1` and `3`, being the first an ordinary boundary with null displacements and the second a crack-like boundary with traction-free faces:

input.dat
```
...

[conditions over be boundaries]
boundary 1: 0 0.
            0 0.
boundary 3: 1 0.
            1 0.
            1 0.
            1 0.

...
```

The same example as before but for a three-dimensional problem:

input.dat
```
...

[conditions over be boundaries]
boundary 1: 0 0.
            0 0.
            0 0.
boundary 3: 1 0.
            1 0.
            1 0.
            1 0.
            1 0.
            1 0.

...
```

Note that if the analysis is time harmonic, the values of the boundary conditions must be introduced as complex numbers, e.g.`(0.,0.)`.

2.6.3 Section `conditions over nodes`

In this section, the boundary conditions over nodes are applied. For nodes of boundary elements, this boundary condition over nodes overrides the boundary conditions applied over their boundaries, and the Tables 2.9 and 2.10 remains valid. For nodes of finite elements, the boundary conditions are applied to the displacements and the rotation, the type of boundary conditions are `0` for known displacement, and `1` for known force (or moment).

The general format for the section is similar to the conditions over boundaries:

general format of section [conditions over nodes]
```
[conditions over nodes]
node <node id>: <type x> <value x>
                <type y> <value y>
                <type z> <value z>
                <if fe: type for rot. x or alpha> <value for rot. x or alpha>
                <if fe: type for rot. y or beta>  <value for rot. y or beta>
                <if fe: type for rot. z>          <value for rotation z>
```

For a two-dimensional problem, example of a clamped beam FE node with identifier `67`, and a pin beam FE node with identifier `41`:

`input.dat`

```
...

[conditions over nodes]
node 67: 0 0.
         0 0.
         0 0.
node 41: 0 0.
         0 0.
         1 0.
```

2.6.4 Section `incident waves`

In this section, the incoming waves are defined. The incidence angles are shown in Figure 2.4.

Figure 2.4: Incidence angles for two- and three-dimensional problems

The incident waves can be defined in terms of displacements (unitary displacements), in terms of stresses (unitary stresses) or in terms of the potentials (unitary potentials). The general format for the section is:

`general format of section [incident waves]`

```
[incident waves]
<number of incident waves>

<incident wave identifier>
<class: plane>
<space> [if space=half-space: <np> <xp> <bc>]
<variable> <amplitude> <x0(1)> <x0(2)> [if 3D: <x0(3)>] [if 3D: <varphi>] <theta>
<xs(1)> <xs(2)> [if 3D: <xs(3)>] <symconf(1)> <symconf(2)> [if 3D: <symconf(3)>]
<region_type> <wave_type>

...
```

where angles have to be introduced in degrees.

Example of vertical P-wave in terms of displacements in the full-space:

`input.dat`

```
...

[incident waves]
1
1
plane
full-space
0 1. 0. 0. 90.
0. 0. 0. 0.
viscoelastic p

...
```

Example of vertical SV-wave in terms of displacements in the half-space:

`input.dat`

```
...

[incident waves]
1
1
plane
half-space 2 0. 1
0 1. 0. 0. 90.
0. 0. 0. 0.
viscoelastic sv

...
```

Chapter 3

Output files

3.1 Introduction

Output files are written according to the type of analysis and model, as well as the input file export section. By default the path and name of the output files is equal to the input file but with an additional extension. The path and name can be changed when calling the solver from the terminal as explained in 1.4. There are three main native output files with the following additional extensions: nodal solutions (`*.nso`), element solutions (`*.eso`) and stress resultant solutions (`*.tot`). The specific file format is different for static and time harmonic analysis. There is one Gmsh output file (`*.pos`) which contains the case mesh and results (MSH file format version 2.2).

3.2 Nodal solutions file (`*.nso`)

This output file is a plain text file containing the nodal (node by node) results. The file starts with a set of header lines whose first character is "#" (a comment line in GNU/Linux environments), describing the file and the meaning of the main data columns. The first 9 columns are always present, and indicates the following:

Column	**Value**	**Description**
$1	integer	Analysis step index. For linear elastic static analysis it is always 0. For time harmonic analysis it is the frequency index. In future releases, it will receive the natural frequency index (for modal analysis), or the time step index (for transient analysis), or the loading step index (for non-linear static analysis).
$2	float	Analysis step value. For linear elastic static analysis it is always 0.0. For time harmonic analysis it is the frequency value (in Hz or rad/s depending on the units used in the frequencies section of the input file). In future releases, it will receive the natural frequency value (for modal analysis), or the time step value (for transient analysis), or the loading step value (for non-linear static analysis).
$3	integer	Identifier of the region to which the node result belongs. In the case of finite element nodes, this has no significance since results does not depend on the region. However, in the case of boundary elements belonging to two regions (interface boundary elements) this allows selecting which result is required.
$4	integer	Region class: 1 (BE region) or 2 (FE region).
$5	integer	Region type: 1 (inviscid fluid) or 2 (elastic solid) or 3 (poroelastic medium).

Column	Value	Description
\$6	integer	Identifier of the BE boundary (if \$4==1) or FE subregion (if \$4==2) to which the node result belongs. In the case of finite element nodes, this has no significance since results does not depend on the FE subregion. However, in the case of boundary elements belonging to two regions (interface boundary elements) this allows selecting which result is required.
\$7	integer	If \$4==1, then it indicates the boundary class: 1 (ordinary) or 2 (crack-like). If \$4==2, then it indicates the number of degrees of freedom of the node.
\$8	integer	If \$4==1, then it indicates the boundary face: 1 (positive normal) or 2 (negative normal). If \$4==2, then it is always 0.
\$9	integer	Identifier of the node.

From column \$10 onwards, the meaning of each column differs depending on the problem dimension (2D or 3D), on the type of analysis (static or time harmonic), on the region class (BE or FE), on the region type (inviscid fluid, elastic solid, poroelastic medium), and on the number of DOF (if a FE region).

For 2D problems in a static analysis (all regions are only of the elastic solid type):

Column	Value	Description
\$10,\$11	float	Node (x_1, x_2) coordinates.
		BE region node
\$12,\$13	float	Node displacements (u_1, u_2).
\$14,\$15	float	Node tractions (t_1, t_2).
		FE region node with 2 DOFs
\$12,\$13	float	Node displacements (u_1, u_2).
\$14,\$15	float	Node forces (f_1, f_2).
		FE region node with 3 DOFs
\$12-\$14	float	Node displacements/rotation (u_1, u_2, θ_3).
\$15-\$17	float	Node forces/moment (f_1, f_2, m_3).

For 3D problems in a static analysis (all regions are only of the elastic solid type):

Column	Value	Description
\$10-\$12	float	Node (x_1, x_2, x_3) coordinates.
		BE region node
\$13-\$15	float	Node displacements (u_1, u_2, u_3).
\$16-\$18	float	Node tractions (t_1, t_2, t_3).
		FE region node with 3 DOFs
\$13-\$15	float	Node displacements (u_1, u_2, u_3).
\$16-\$18	float	Node forces (f_1, f_2, f_3).
		FE region node with 5 DOFs (shell element nodes with local rotations)
\$13-\$17	float	Node displacements/local rotations $(u_1, u_2, u_3, \alpha, \beta)$.
\$18-\$22	float	Node forces/local moments $(f_1, f_2, f_3, m_\alpha, m_\beta)$.

Column	Value	Description
		FE region node with 6 DOFs
\$13-\$18	float	Node displacements/rotations $(u_1, u_2, u_3, \theta_1, \theta_2, \theta_3)$.
\$19-\$24	float	Node forces/moments $(f_1, f_2, f_3, m_1, m_2, m_3)$.

For 2D problems in a time harmonic analysis, where each node variable is written by two consecutive columns containing its real and imaginary parts (if `complex_notation = cartesian` in `[export]` section) or absolute value and argument (if `complex_notation = polar` in `[export]` section):

Column	Value	Description
\$10,\$11	float	Node (x_1, x_2) coordinates.
		BE region node (inviscid fluid)
\$12,\$13	float	Node pressure p (total field).
\$14,\$15	float	Node normal displacement U_n (total field).
\$16,\$17	float	Node pressure p^{inc} (incident field).
\$18,\$19	float	Node normal displacement U_n^{inc} (incident field).
\$20-\$23	float	Node displacement (U_1, U_2) (total field).
		BE region node (elastic solid)
\$12-\$15	float	Node displacements (u_1, u_2) (total field).
\$16-\$19	float	Node tractions (t_1, t_2) (total field).
\$20-\$23	float	Node displacements $(u_1^{\text{inc}}, u_2^{\text{inc}})$ (incident field).
\$24-\$27	float	Node tractions $(t_1^{\text{inc}}, t_2^{\text{inc}})$ (incident field).
		BE region node (poroelastic medium)
\$12,\$13	float	Node fluid equivalent pressure τ (total field). Note that it is related to the dynamic pore pressure by $\tau = -\phi p$, where ϕ is the porosity.
\$14-\$17	float	Node solid displacements (u_1, u_2) (total field).
\$18,\$19	float	Node fluid normal displacement U_n (total field).
\$20-\$23	float	Node solid tractions (t_1, t_2) (total field).
\$24,\$25	float	Node fluid equivalent pressure τ^{inc} (incident field).
\$26-\$29	float	Node solid displacements $(u_1^{\text{inc}}, u_2^{\text{inc}})$ (incident field).
\$30,\$31	float	Node fluid normal displacement U_n^{inc} (incident field).
\$32-\$35	float	Node solid tractions $(t_1^{\text{inc}}, t_2^{\text{inc}})$ (incident field).
\$36-\$39	float	Node fluid displacement (U_1, U_2) (total field).
		FE region node (only of elastic solid type) with 2 DOFs
\$12-\$15	float	Node displacements (u_1, u_2).
\$16-\$19	float	Node forces (f_1, f_2).
		FE region node (only of elastic solid type) with 3 DOFs
\$12-\$17	float	Node displacements/rotation (u_1, u_2, θ_3).
\$18-\$23	float	Node forces/moment (f_1, f_2, m_3).

For 3D problems in a time harmonic analysis, where as in the 2D case each node variable is written by two consecutive columns containing its real and imaginary parts (if `complex_notation = cartesian` in `[export]` section) or absolute value and argument (if `complex_notation = polar` in `[export]` section):

Column	Value	Description
\$10-\$12	float	Node (x_1, x_2, x_3) coordinates.
		BE region node (inviscid fluid)
\$13,\$14	float	Node pressure p (total field).
\$15,\$16	float	Node normal displacement U_n (total field).
\$17,\$18	float	Node pressure p^{inc} (incident field).
\$19,\$20	float	Node normal displacement U_n^{inc} (incident field).
\$21-\$24	float	Node displacement (U_1, U_2, U_3) (total field).
		BE region node (elastic solid)
\$13-\$18	float	Node displacements (u_1, u_2, u_3) (total field).
\$19-\$24	float	Node tractions (t_1, t_2, t_3) (total field).
\$25-\$30	float	Node displacements $(u_1^{\text{inc}}, u_2^{\text{inc}}, u_3^{\text{inc}})$ (incident field).
\$31-\$36	float	Node tractions $(t_1^{\text{inc}}, t_2^{\text{inc}}, t_3^{\text{inc}})$ (incident field).
		BE region node (poroelastic medium)
\$13,\$14	float	Node fluid equivalent pressure τ (total field). Note that it is related to the dynamic pore pressure by $\tau = -\phi p$, where ϕ is the porosity.
\$15-\$20	float	Node solid displacements (u_1, u_2, u_3) (total field).
\$21,\$22	float	Node fluid normal displacement U_n (total field).
\$23-\$28	float	Node solid tractions (t_1, t_2, t_3) (total field).
\$29,\$30	float	Node fluid equivalent pressure τ^{inc} (incident field).
\$31-\$36	float	Node solid displacements $(u_1^{\text{inc}}, u_2^{\text{inc}}, u_3^{\text{inc}})$ (incident field).
\$37,\$38	float	Node fluid normal displacement U_n^{inc} (incident field).
\$39-\$44	float	Node solid tractions $(t_1^{\text{inc}}, t_2^{\text{inc}}, t_3^{\text{inc}})$ (incident field).
\$45-\$50	float	Node fluid displacement (U_1, U_2, U_3) (total field).
		FE region node (only of elastic solid type) with 3 DOFs
\$13-\$18	float	Node displacements (u_1, u_2, u_3).
\$19-\$24	float	Node forces (f_1, f_2, f_3).
		FE region node (only of elastic solid type) with 5 DOFs (shell element nodes with local rotations)
\$13-\$22	float	Node displacements/local rotations $(u_1, u_2, u_3, \alpha, \beta)$.
\$23-\$32	float	Node forces/local moments $(f_1, f_2, f_3, m_\alpha, m_\beta)$.
		FE region node (only of elastic solid type) with 6 DOFs
\$13-\$24	float	Node displacements/rotation $(u_1, u_2, u_3, \theta_1, \theta_2, \theta_3)$.
\$25-\$36	float	Node forces/moment $(f_1, f_2, f_3, m_1, m_2, m_3)$.

3.3 Element solutions file (`*.eso`)

This output file is a plain text file containing the element (element node by element node) results (stress resultants on finite elements). In a future release, element by element stress resultants over each boundary element will also be included. The file starts with a set of header lines whose first character is “`#`” (a comment line in GNU/Linux environments), describing the file and the meaning of the main data columns. The first 13 columns are always present, and indicates the following:

Column	**Value**	**Description**
\$1	integer	Analysis step index. For linear elastic static analysis it is always 0. For time harmonic analysis it is the frequency index. In future releases, it will receive the natural frequency index (for modal analysis), or the time step index (for transient analysis), or the loading step index (for non-linear static analysis).
\$2	float	Analysis step value. For linear elastic static analysis it is always 0.0. For time harmonic analysis it is the frequency value (in Hz or rad/s depending on the units used in the frequencies section of the input file). In future releases, it will receive the natural frequency value (for modal analysis), or the time step value (for transient analysis), or the loading step value (for non-linear static analysis).
\$3	integer	Identifier of the region to which the element node result belongs. In the case of finite element nodes, this has no significance since results does not depend on the region. However, in the case of boundary elements belonging to two regions (interface boundary elements) this allows selecting which result is required.
\$4	integer	Region class: 1 (BE region) or 2 (FE region).
\$5	integer	Region type: 1 (inviscid fluid) or 2 (elastic solid) or 3 (poroelastic medium).
\$6	integer	Identifier of the BE boundary (if \$4==1) or FE subregion (if \$4==2) to which the element result belongs. In the case of boundary elements belonging to two regions (interface boundary elements) this allows selecting which result is required.
\$7	integer	If \$4==1, then it indicates the boundary class: 1 (ordinary) or 2 (crack-like). If \$4==2, then it indicates the finite element dimension.
\$8	integer	If \$4==1, then it indicates the boundary face: 1 (positive normal) or 2 (negative normal). If \$4==2, then it indicates the finite element type.
\$9-\$11	integer	Element identifier, dimension and type.
\$12,\$13	integer	Element node index and number of degrees of freedom.
\$14	integer	Node identifier.

For 2D problems in a static analysis:

Column	**Value**	**Description**
\$15,\$16	float	Node (x_1, x_2) coordinates.
		Straight beam finite element
\$17-\$19	float	Axial force ($N_{x'}$), shear force ($V_{y'}$) and bending moment ($M_{z'}$).
		Bar finite element
\$17	float	Axial force ($N_{x'}$).
		Discrete translational spring finite element
\$17,\$18	float	Force on each spring: $N_{x'}$, $N_{y'}$.
		Discrete translational/rotational spring finite element
\$17-\$19	float	Force/moment on each spring: $N_{x'}$, $N_{y'}$, $M_{z'}$.

For 3D problems in a static analysis:

Column	Value	Description
\$15-\$17	float	Node (x_1, x_2, x_3) coordinates.
		Straight beam finite element
\$18-\$23	float	Axial force ($N_{x'}$), shear forces ($V_{y'}$, $V_{z'}$), torsional moment ($M_{z'}$) and bending moments ($M_{y'}$, $M_{z'}$).
		Bar finite element
\$18	float	Axial force ($N_{x'}$).
		Discrete translational spring finite element
\$18-\$20	float	Force on each spring: $N_{x'}$, $N_{y'}$, $N_{z'}$.
		Discrete translational/rotational spring finite element
\$18-\$23	float	Force/moment on each spring: $N_{x'}$, $N_{y'}$, $N_{z'}$, $M_{x'}$, $M_{y'}$, $M_{z'}$.
		Discrete translational/rotational spring finite element
\$18-\$25	float	Membrane forces ($N_{x'}$, $N_{y'}$, $N_{x'y'}$), bending moments ($M_{x'}$, $M_{y'}$, $M_{x'y'}$) and shear forces ($V_{x'}$, $V_{y'}$).

For 2D problems in a time harmonic analysis, where each node variable is written by two consecutive columns containing its real and imaginary parts (if `complex_notation = cartesian` in `[export]` section) or absolute value and argument (if `complex_notation = polar` in `[export]` section):

Column	Value	Description
\$15,\$16	float	Node (x_1, x_2) coordinates.
		Straight beam finite element
\$17-\$22	float	Axial force ($N_{x'}$), shear force ($V_{y'}$) and bending moment ($M_{z'}$).
		Bar finite element
\$17,\$18	float	Axial force ($N_{x'}$).
		Discrete translational spring finite element
\$17-\$20	float	Force on each spring: $N_{x'}$, $N_{y'}$.
		Discrete translational/rotational spring finite element
\$17-\$22	float	Force/moment on each spring: $N_{x'}$, $N_{y'}$, $M_{z'}$.

For 3D problems in a time harmonic analysis, where each node variable is written by two consecutive columns containing its real and imaginary parts (if `complex_notation = cartesian` in `[export]` section) or absolute value and argument (if `complex_notation = polar` in `[export]` section):

Column	Value	Description
\$15-\$17	float	Node (x_1, x_2, x_3) coordinates.
		Straight beam finite element
\$18-\$29	float	Axial force ($N_{x'}$), shear forces ($V_{y'}$, $V_{z'}$), torsional moment ($M_{z'}$) and bending moments ($M_{y'}$, $M_{z'}$).
		Bar finite element

Column	Value	Description
\$18,\$19	float	Axial force ($N_{x'}$).
		Discrete translational spring finite element
\$18-\$23	float	Force on each spring: $N_{x'}$, $N_{y'}$, $N_{z'}$.
		Discrete translational/rotational spring finite element
\$18-\$29	float	Force/moment on each spring: $N_{x'}$, $N_{y'}$, $N_{z'}$, $M_{x'}$, $M_{y'}$, $M_{z'}$.
		Discrete translational/rotational spring finite element
\$18-\$33	float	Membrane forces ($N_{x'}$, $N_{y'}$, $N_{x'y'}$), bending moments ($M_{x'}$, $M_{y'}$, $M_{x'y'}$) and shear forces ($V_{x'}$, $V_{y'}$).

3.4 Gmsh results file (`*.pos`)

The Gmsh output file (`*.pos`) contains the case mesh and results (MSH file format version 2.2). It is a plain text containing all this data which can be read by Gmsh. The following set of results are exported to Gmsh:

- Fluid pressure (positive faces). Only boundary elements.
- Fluid pressure (negative faces). Only boundary elements.
- Solid total displacements (positive faces). Boundary elements and finite elements.
- Solid total displacements (negative faces). Only boundary elements.
- Solid total tractions (positive faces). Only boundary elements.
- Solid total tractions (negative faces). Only boundary elements.
- FE nodal forces/reactions. Only finite elements.
- Beam stress resultants. Only finite elements.
- Beam stress resultants local axis x', y' and z'. Only finite elements.
- Bar stress resultants. Only finite elements.
- Shell stress resultants. Only finite elements.
- Shell stress resultants local axis x', y' and z'. Only finite elements.

Appendix A

How to compile the source code

Go to MultiFEBE's GitHub webpage, and download the source code. Then, decompress the `*.tar.gz` or `*.zip` file.

A.1 GNU/Linux

The steps to compile the source code are the usual on GNU/Linux projects using CMake. The particular commands shown below are for Debian distributions (e.g. Ubuntu), but it should be very similar for others.

1. First, you have to install the pre-requisites if not already installed, so you have to open a terminal and execute:

   - Install GNU Fortran, GNU Make and CMake:

     ```
     $ sudo apt-get install gfortran make cmake cmake-extras
     ```

   - Install OpenBLAS:

     ```
     $ sudo apt-get install libopenblas-base libopenblas-dev
     ```

2. Once you have all the pre-requisites, navigate at the source code main folder. Then create a temporary folder (e.g. `build`) where making the compilation, and navigate inside it:

   ```
   $ mkdir -p build
   $ cd build
   ```

3. Execute cmake taking into account that the CMake configuration file is in the parent folder:

   ```
   $ cmake -G "Unix Makefiles" ..
   ```

   During the process, if any tool or dependency is missing, then CMake is going to notify you.

4. Once CMake correctly finishes, you can now execute GNU Make:

   ```
   $ make
   ```

5. If all this process ends correctly, you will have the executable `multifebe` there. However, it is still not available system-wide in the terminal. You have at this point several options:

   (a) The simpler one is to copy the binary to `usr/bin`:

   ```
   $ sudo cp multifebe /usr/bin/.
   ```

   (b) The cleaner one is making a `*.deb` installer which will manage not only this, but it will also copy the documentation to the corresponding folders (`/usr/share/doc/multifebe`), and create a package registry so that it can also be completely uninstalled or updated in the future. In order to do so, you have to execute CPack as:

   ```
   $ cpack
   ```

   At the end of the process, you will have at your disposal your own `*.deb` installer. Then, you can follow the step described above in section 1.3.1.

A.2 Windows

The compilation in Windows requires much more steps than in GNU/Linux, but basically because it is required to install and to configure MSYS2. MSYS2 is a collection of tools and libraries providing you with an easy-to-use environment for building, installing and running native Windows software. Furthermore, it is very similar to GNU/Linux environments, so most of the programming work is shared when building for GNU/Linux and Windows.

1. Download and install MSYS2 from the webpage.

2. After the installation, the ”MSYS2 MSYS” terminal is automatically opened. Close it.

3. We recommend to follow the post-installation instructions given at the webpage.

4. MSYS2 installs the so called Mintty terminal, but it provides separate launchers for different environments, such as the already opened “MSYS2 MSYS”, but also “MinGW x64”, which is the one used for compiling, and others. See Figure A.1.

5. We will reproduce the post-installation instructions given by MSYS2 at the date of this writing. First, open the “MSYS2 MSYS” terminal. See Figure A.1.

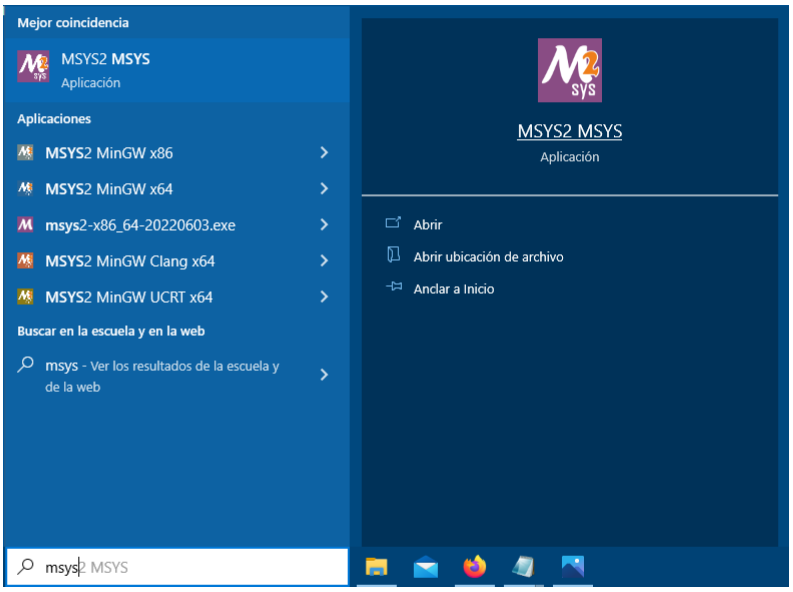
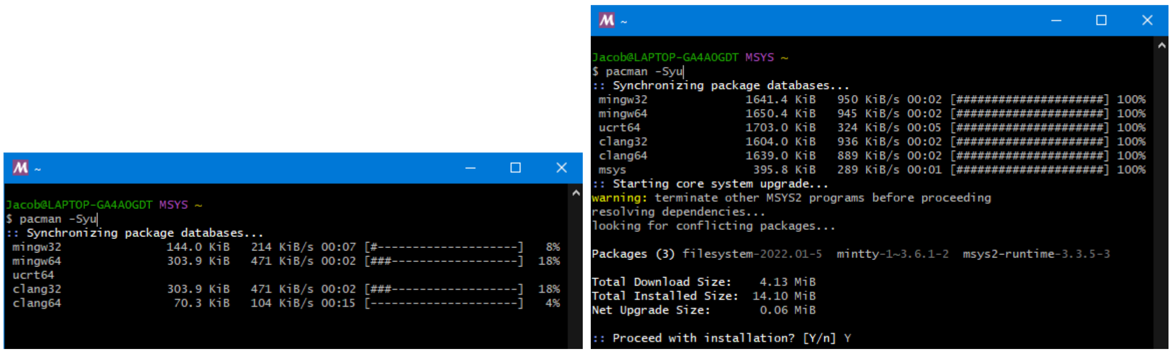


Figure A.1: Compilation in Windows: Steps 4 and 5

6. Update the packages by typing and executing the command `pacman -Syu` as shown in Figure A.3.



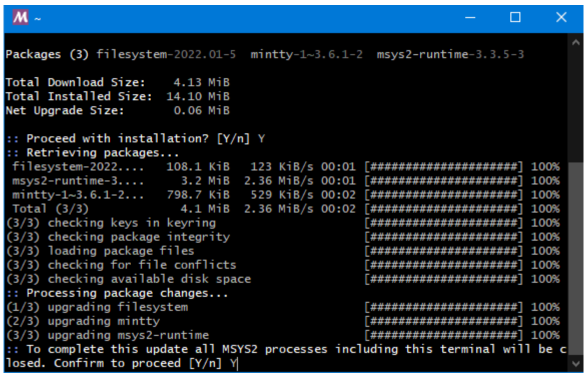
```
Jacob@LAPTOP-GA4A0GDT MSYS ~
$ pacman -Syu
:: Synchronizing package databases...
 mingw32                  144.0 KiB   214 KiB/s 00:07 [#--------------------]   8%
 mingw64                  303.9 KiB   471 KiB/s 00:02 [###------------------]  18%
 ucrt64
 clang32                  303.9 KiB   471 KiB/s 00:02 [###------------------]  18%
 clang64                   70.3 KiB   104 KiB/s 00:15 [---------------------]   4%
```

```
Jacob@LAPTOP-GA4A0GDT MSYS ~
$ pacman -Syu
:: Synchronizing package databases...
 mingw32             1641.4 KiB   950 KiB/s 00:02 [#####################] 100%
 mingw64             1650.4 KiB   945 KiB/s 00:02 [#####################] 100%
 ucrt64              1703.0 KiB   324 KiB/s 00:05 [#####################] 100%
 clang32             1604.0 KiB   936 KiB/s 00:02 [#####################] 100%
 clang64             1639.0 KiB   889 KiB/s 00:02 [#####################] 100%
 msys                 395.8 KiB   289 KiB/s 00:01 [#####################] 100%
:: Starting core system upgrade...
warning: terminate other MSYS2 programs before proceeding
resolving dependencies...
looking for conflicting packages...

Packages (3) filesystem-2022.01-5  mintty-1~3.6.1-2  msys2-runtime-3.3.5-3

Total Download Size:    4.13 MiB
Total Installed Size:  14.10 MiB
Net Upgrade Size:       0.06 MiB

:: Proceed with installation? [Y/n] Y
```

Figure A.2: Compilation in Windows: Steps 5 and 6

7. Proceed with the installation by typing `Y` and executing as shown in Figure A.3.

8. After updating, the terminal ask permission for closing. Type `Y` and execute.



```
Packages (3) filesystem-2022.01-5  mintty-1~3.6.1-2  msys2-runtime-3.3.5-3

Total Download Size:    4.13 MiB
Total Installed Size:  14.10 MiB
Net Upgrade Size:       0.06 MiB

:: Proceed with installation? [Y/n] Y
:: Retrieving packages...
 filesystem-2022....   108.1 KiB   123 KiB/s 00:01 [#####################] 100%
 msys2-runtime-3....     3.2 MiB  2.36 MiB/s 00:01 [#####################] 100%
 mintty-1~3.6.1-2...   798.7 KiB   529 KiB/s 00:02 [#####################] 100%
 Total (3/3)             4.1 MiB  2.36 MiB/s 00:02 [#####################] 100%
(3/3) checking keys in keyring                     [#####################] 100%
(3/3) checking package integrity                   [#####################] 100%
(3/3) loading package files                        [#####################] 100%
(3/3) checking for file conflicts                  [#####################] 100%
(3/3) checking available disk space                [#####################] 100%
:: Processing package changes...
(1/3) upgrading filesystem                         [#####################] 100%
(2/3) upgrading mintty                             [#####################] 100%
(3/3) upgrading msys2-runtime                      [#####################] 100%
:: To complete this update all MSYS2 processes including this terminal will be c
losed. Confirm to proceed [Y/n] Y
```

Figure A.3: Compilation in Windows: Step 8

9. Open again the "MSYS2 MSYS" terminal, and type and execute again the update command `pacman -Syu`.
10. Proceed with the installation by typing and executing `Y`.
11. Install typical tools for compiling by typing and executing:

    ```
    $ pacman -S mingw-w64-x86_64-toolchain
    ```

    which will take some time.
12. Close the "MSYS2 MSYS" terminal.
13. Open the "MSYS2 MinGW x64" terminal.

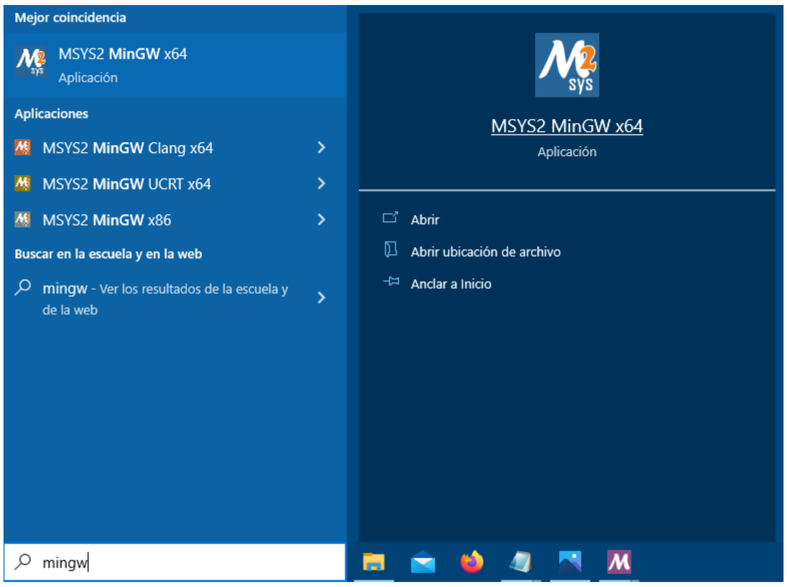


Figure A.4: Compilation in Windows: Step 13

14. Install additional tools and libraries required by typing and executing the following sequence of commands:

    (a) Install GNU C Compiler and GNU Fortran for MinGW 64 bits:

    ```
    $ pacman -S gcc mingw-w64-x86_64-gcc gcc-fortran mingw-w64-x86_64-gcc-fortran
    ```

    (b) Install GNU Make, CMake and NSIS:

    ```
    $ pacman -S gcc make mingw-w64-x86_64-make cmake mingw-w64-x86_64-cmake mingw-w64-x86_64-nsis
    ```

    (c) Install OpenBLAS library:

```
$ pacman -S gcc mingw-w64-x86_64-openblas
```

15. In the "MSYS2 MinGW x64" terminal, navigate to the source code main folder. In the case illustrated in the following figures `D:\multifebe`.

16. Create a folder where all compilation build files are generated. In the case illustrated in the following figures it is named `build`. This can be done by executing:

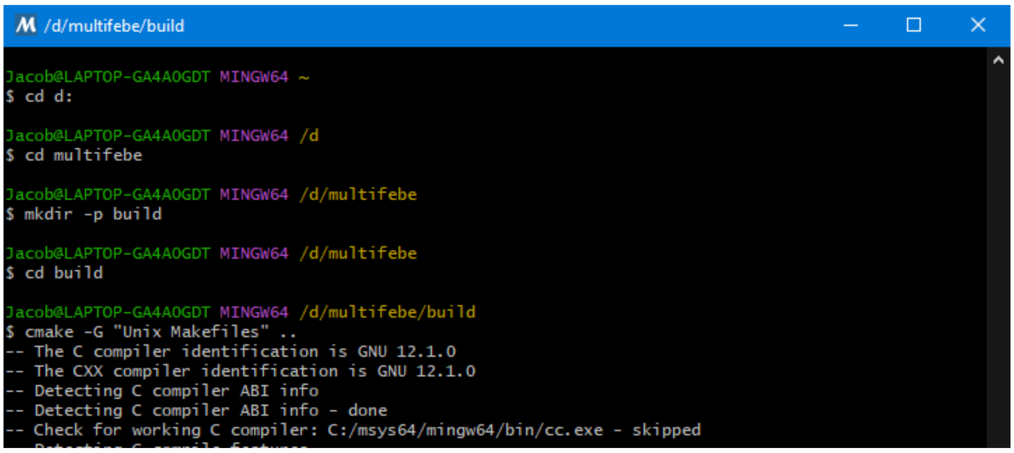
```
$ mkdir -p build
```

17. Navigate to it by executing:

```
$ cd build
```

18. Execute `cmake` taking into account that we want to use make as the compilation tool (thus using the option `-G ``Unix Makefiles''`) and that the configuration file `CMakeLists.txt` is in the parent folder:

```
$ cmake -G "Unix Makefiles" ..
```



```
Jacob@LAPTOP-GA4A0GDT MINGW64 ~
$ cd d:

Jacob@LAPTOP-GA4A0GDT MINGW64 /d
$ cd multifebe

Jacob@LAPTOP-GA4A0GDT MINGW64 /d/multifebe
$ mkdir -p build

Jacob@LAPTOP-GA4A0GDT MINGW64 /d/multifebe
$ cd build

Jacob@LAPTOP-GA4A0GDT MINGW64 /d/multifebe/build
$ cmake -G "Unix Makefiles" ..
-- The C compiler identification is GNU 12.1.0
-- The CXX compiler identification is GNU 12.1.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/msys64/mingw64/bin/cc.exe - skipped
```

Figure A.5: Compilation in Windows: Steps 15 to 18

19. Execute `make` for compiling the source code:

```
$ make
```

20. After this is correctly done, `multifebe.exe` should be ready to be used in the current folder.

21. You can manually install this program in the system by creating a new folder inside `C:\Files (x86)` called for example `multifebe`, and copy `multifebe.exe` inside it. Then you have to add this directory to the system or user `PATH` environment variable. To do so, follow the instructions in this link, or do a search on the web about "How to set the path and environment variables in Windows". After this is done, you can run the program on a system terminal, like cmd or PowerShell.

22. It is also possible to build an installer which do all this for you, and furthermore integrates the program into your system with a uninstaller, etc. You have to be in a "MSYS2 MinGW x64" terminal, and the current working directory in the folder where we are compilating.

23. Then you can build the installer by executing in the terminal:

```
$ cpack
```

24. This will create and installer called `multifebe-x.x.x-w64.exe`, which is the installer we release for Windows. Then you can execute it and follow the instructions above in section 1.3.3.

Appendix B

GiD *.bas template file

We have implemented a *.bas template file for writing mesh files in the MultiFEBE native format from GiD pre- and post-processor. The carriage return must be of the Windows or the Unix type depending on which OS is GiD running. The shown blank lines are necessary. It is important to note that each GiD “layer” is a “part” in MultiFEBE jargon. This file is also included in the package.

multifebe.bas

```

*IntFormat "%12i"
*RealFormat "%25.16e"
[parts]
*Set var nLayer=0
*loop layers
*Set var nLayer=LayerNum
*end layers
*nLayer
*loop layers
*LayerNum *LayerName
*end layers
[nodes]
*nPoin
*set elems(all)
*loop nodes
*NodesNum *NodesCoord
*end nodes
[elements]
*set Elems(All)
*nElem
*set Elems(Linear)
*loop elems
*if(ElemsNnode==2)
*ElemsNum line2 1 *ElemsLayerNum *ElemsConec
*endif
*if(ElemsNnode==3)
*ElemsNum line3 1 *ElemsLayerNum *ElemsConec
*endif
*end elems
*set Elems(Triangle)
*loop elems
*if(ElemsNnode==3)
*ElemsNum tri3 1 *ElemsLayerNum *ElemsConec
*endif
*if(ElemsNnode==6)
*ElemsNum tri6 1 *ElemsLayerNum *ElemsConec
*endif
*end elems
*set Elems(Quadrilateral)
*loop elems
*if(ElemsNnode==4)
*ElemsNum quad4 1 *ElemsLayerNum *ElemsConec
*endif
*if(ElemsNnode==8)
*ElemsNum quad8 1 *ElemsLayerNum *ElemsConec
*endif
*if(ElemsNnode==9)
*ElemsNum quad9 1 *ElemsLayerNum *ElemsConec
*endif
*end elems

```

Bibliography

[1] O. Maeso, J. J. Aznárez, and J. Domínguez. Three-dimensional models of reservoir sediment and effects on the seismic response of arch dams. *Earthquake Engineering and Structural Dynamics*, 33(10):1103–1123, 2004.

[2] J. J. Aznárez, O. Maeso, and J. Domínguez. BE analysis of bottom sediments in dynamic fluid-structure interaction problems. *Engineering Analysis with Boundary Elements*, 30:124–136, 2006.

[3] L.A. Padrón, J.J. Aznárez, and O. Maeso. BEM–FEM coupling model for the dynamic analysis of piles and pile groups. *Engineering Analysis with Boundary Elements*, 31(6):473 – 484, 2007.

[4] Guillermo M. Álamo, Alejandro E. Martínez-Castro, Luis A. Padrón, Juan J. Aznárez, Rafael Gallego, and Orlando Maeso. Efficient numerical model for the computation of impedance functions of inclined pile groups in layered soils. *Engineering Structures*, 126:379–390, 2016.

[5] J. D. R. Bordón, J. J. Aznárez, and O. Maeso. Dynamic model of open shell structures buried in poroelastic soils. *Computational Mechanics*, 60(2):269–288, 2017.

[6] J. Domínguez. *Boundary Elements in Dynamics.* WIT Press, 1993.

[7] M.A. Biot. Theory of propagation of elastic waves in a fluid-saturated porous solid. i. low-frequency range. *The Journal of the Acoustical Society of America*, 28(168), 1956.