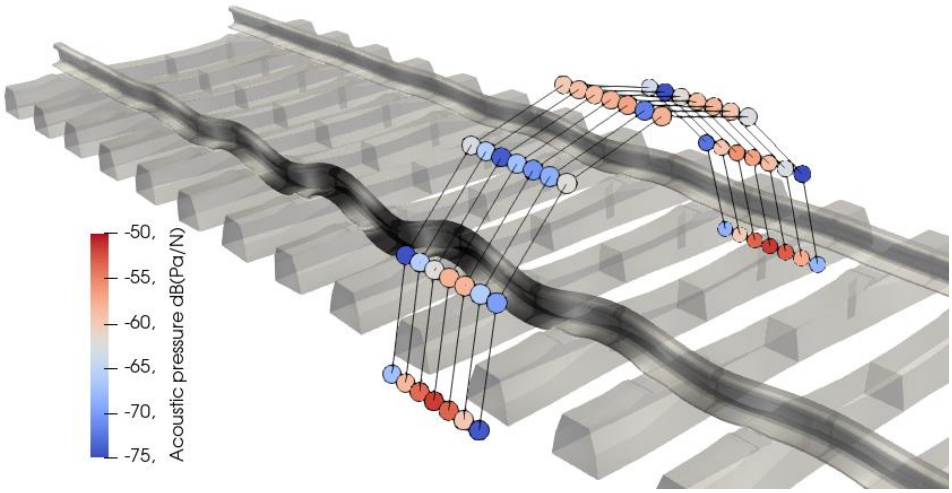
Rail Track Modelling Toolbox

Multi-Sleeper Model

User Guide & Developer Guide



Heig-VD
Raphaël Nardin
March 2024

Table of contents

1. Introduction

1.1. Context

The Rail Track Modelling Toolbox is an open source tool developed at the University of Applied Sciences (Heig-VD, Yverdon-les-Bains) throughout several projects commissioned by the Swiss Federal Office for Environment (FOEN) and in partnership with the Swiss Federal Railways (SBB). It allows performing numerical simulations of rail track components. This document focuses on one of the main models of the toolbox: the Multi-Sleeper Model.

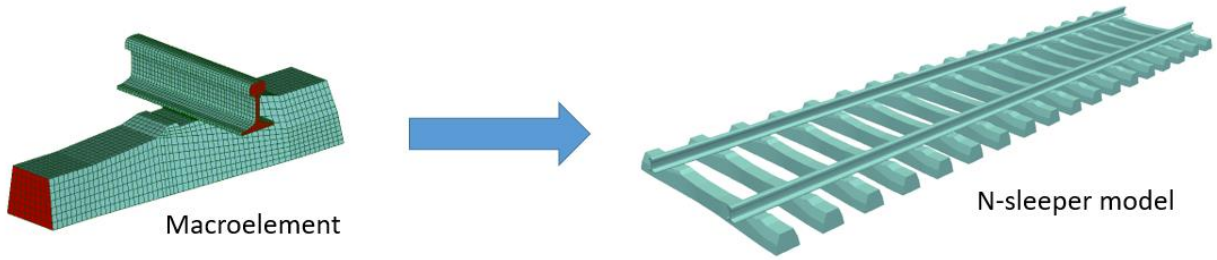
1.2. Model description

The Multi-Sleeper Model performs vibro-acoustic finite element (FE) simulations of large-scale rail tracks superstructure. It includes the rails, the rail pads, the sleepers, the clamps and the ballast (modelled by discrete springs). The harmonic simulations allow making track dynamics predictions and the results were validated experimentally up to 2kHz in terms of track decay rate (TDR), rail vibration and sleeper vibration. The model is also used to evaluate the acoustic pressure at some locations as sensors would do, or to compute the acoustic power radiated through a surface. The acoustic part was not validated in an absolute way, however variations between different systems are well predicted.

Simulations are executed with the open source FE solver Code_Aster (www.code-aster.org) and pre and post-processed with SALOME (www.salome-platform.org). Some modelling techniques were implemented to make the model computationally very efficient to simulate such large models.

1. **Dynamic substructuring:** a macroelement consisting of one half sleeper, one rail pad, a portion of rail, the clamps and eventually an under-sleeper pad (USP) is replicated 2N times, N being the number of sleepers, to make the whole track. Using an appropriate modal basis, the full-scale simulation is much quicker.
2. **Eigenmodes pre-calculation and reuse:** during a first phase, the eigenmodes of the macroelement are computed. The large modal basis is reduced to a small set of modes comprising the most relevant ones, and used as an input in the second phase (*i.e.* the main simulation: harmonic). In that phase, the large stiffness and mass matrices of the actual macroelement are projected on the reduced pre-computed modal basis to enable computing eigenmodes very quickly (QZ method).
3. **Parallelization:** during the main simulation, multiple jobs are run simultaneously, each job simulating a different part of the frequency spectrum defined by the user.

Finally, the acoustic part is based on a monopole superposition technique and is usually the most time-consuming part. A set of radiant surfaces is considered to act as monopoles and all pressure contributions are summed to get the total pressure at a location. The process is repeated for all requested locations.



2. User Guide

In addition to this User Guide, note that nearly all text boxes, buttons and other GUI components have tool tips. Simply place your mouse on any of them and explanations will pop up to guide you through the parameters of the model.

2.1. Toolbox setup and GUI launcher

Once the Toolbox folder is on the Linux machine and that all required libraries are installed (see Toolbox documentation), the first thing to do is to execute the script ./setup0.sh in a teminal. It converts all Windows line ends to Unix line ends in all files in case there are some.

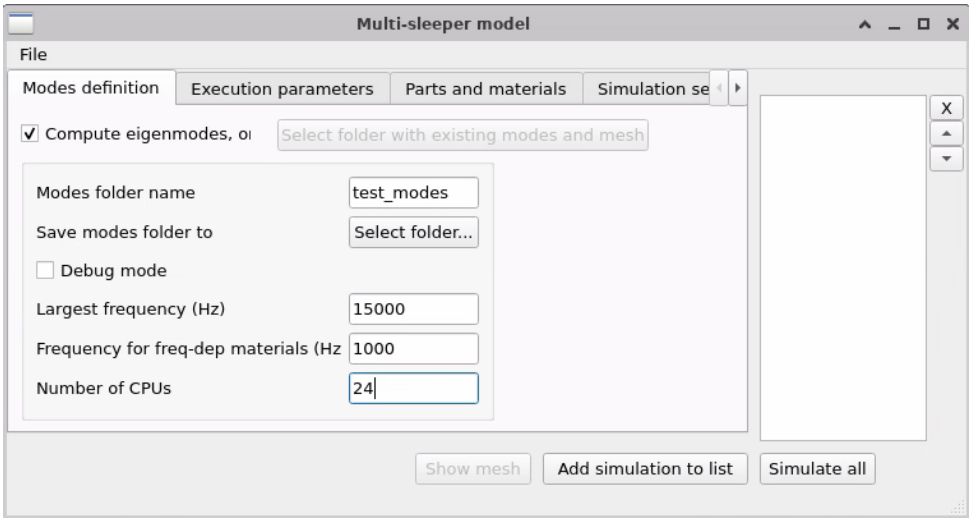
Then, the script ./setup.sh should be opened to check the paths of “python”, “Salome” and “as_run” commands. After updating them if necessary, ./setup.sh can be executed in a terminal. It will display which files will be updated with correct paths and the user must confirm by entering “y” for yes.

The Rail Track Modelling Toolbox can be run in a terminal with the script ./src/RailTrackToolboxLauncher/RailTrackToolboxLauncher.sh. It opens a GUI from which the GUIs of all models can be opened. The GUI of the Multi-Sleeper Model can also be run separately by executing the script ./src/MultiSleeperModel/MultiSleeperModel.sh, which is equivalent.

2.2. Phase 1

Under the *Modes definition* tab, select either to compute the eigenmodes of the macroelement, or to reuse eigenmodes that already exist. In the first case, eigenmodes and information will be saved in a directory named *test_modes* in this example, located in the directory selected below. In the second case, browse the corresponding folder. The condition is that the meshes defined here must correspond to the mesh that was used to pre-compute the modes selected here.

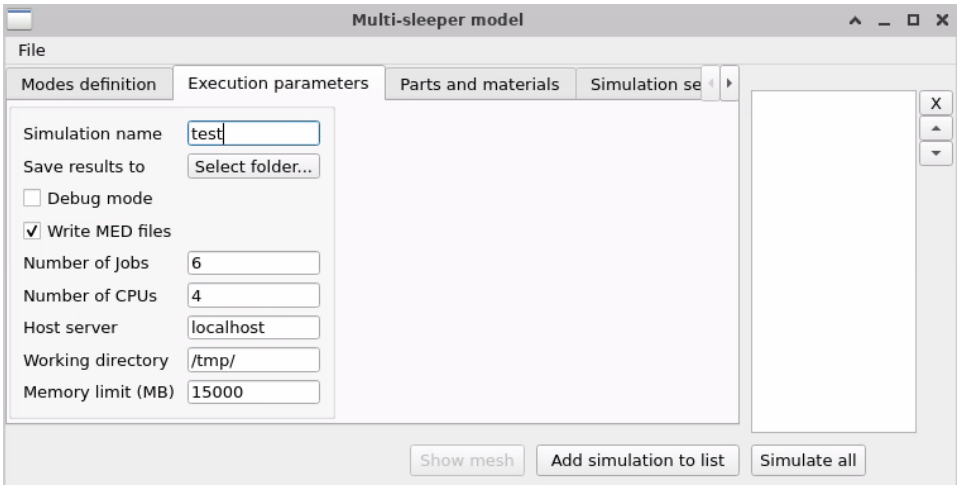
Then, enter the desired values to all parameters, whose explanations are available in the tool tips.



2.3. Execution parameters

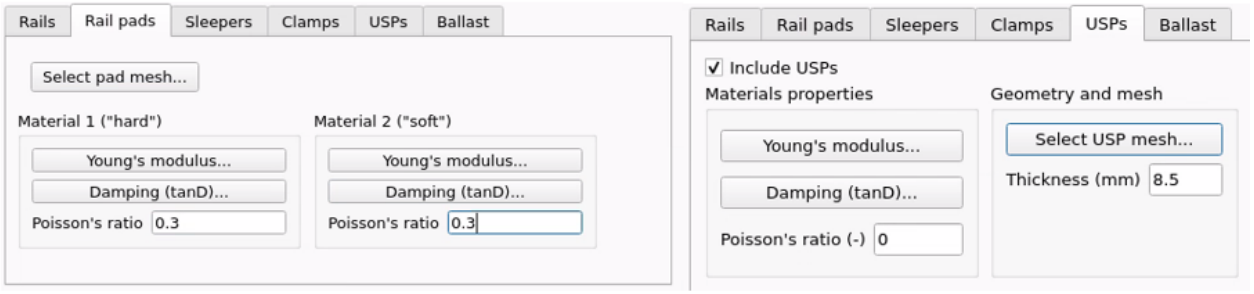
In the *Execution parameters* tab, enter a simulation name and a folder. The results will be saved to *test*, in this example, in the folder selected below. Enter the other parameters by taking into account the performances of your machine. If needed, read carefully the tool tips of the GUI widgets.

Concerning the numbers of jobs and CPUs: the number of jobs defines in how many bands the frequency spectrum (defined later) is divided. The more jobs, the better frequency-dependent materials properties are modelled. Moreover, each job will simulate less frequency values, which can considerably speed up simulations that include acoustic calculation, which takes a lot of time for each frequency. However, the number of jobs times the number of CPU per job must not exceed the total number of CPUs available. Hence, for a heavy acoustic simulation, it is recommended to use as many jobs as possible, and for harmonic simulations without acoustic part, there is an optimum to guess. Usually, 6 jobs times 4 CPUs is a good trade-off, for example if your machine has 24 CPUs.



2.4. Parts and materials

Select the mesh and/or properties of each component: rail, rail pad, sleeper, clamp, USP (optional) and ballast. Refer to the Developer Guide for detailed mesh (3.4 Meshes) and materials (3.5 Materials modelling) information. Refer to the tool tips in the GUI as well.

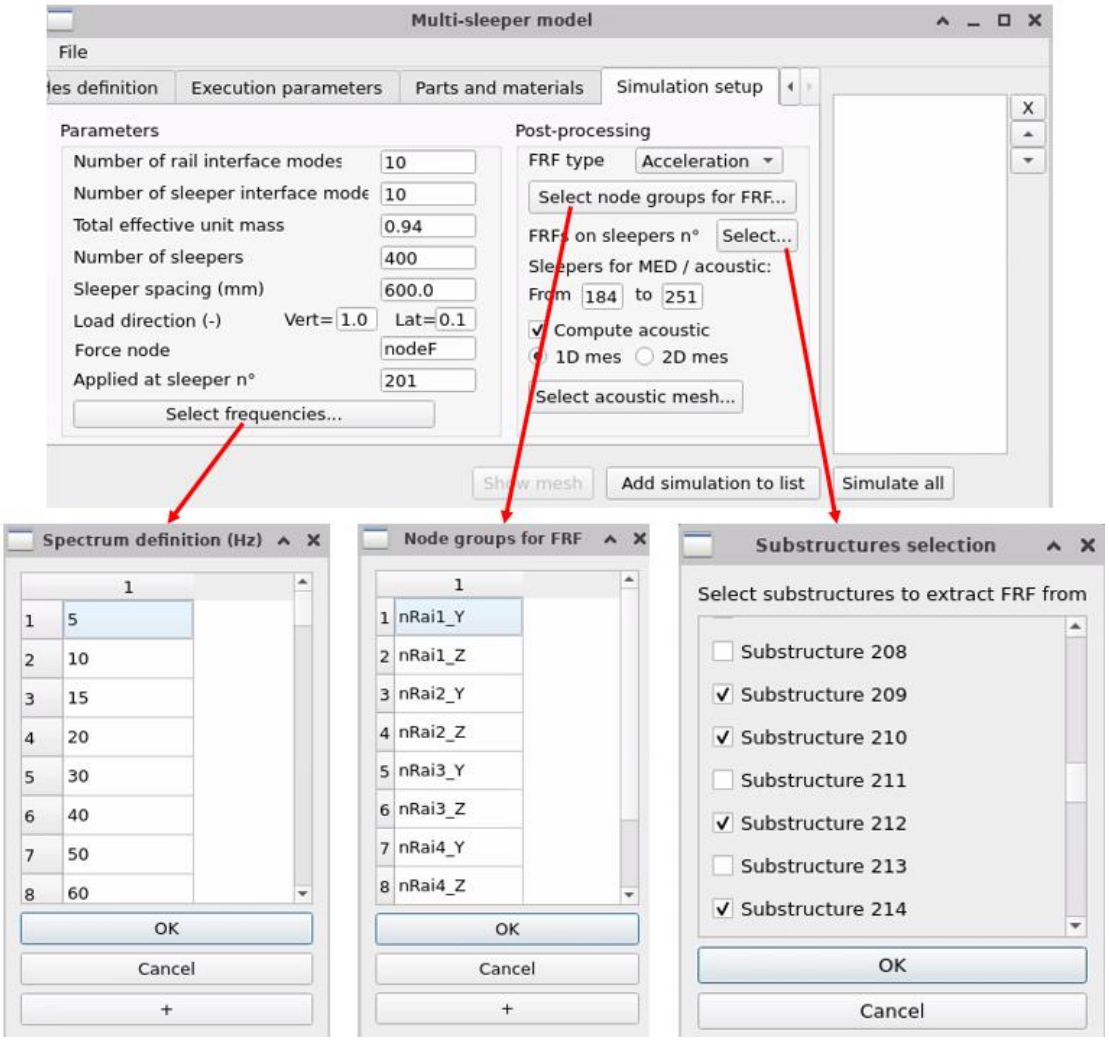


Notes: frequency-dependent CSV files contain frequencies (Hz) in the first column and associated storage modulus (MPa) or damping ratio (-) in the second column. For rail pads, two materials must be defined. They can be identical, but refer respectively to the “hard” and to the “soft” groups of elements. For sleepers, the height from the bottom to the middle of the face in contact with the rail pad must be provided. The default values are 214mm for B91 sleepers and 150mm for RpIV wooden sleepers. For USP, the thickness must be provided as well. The default value for the USP associated to B91 sleepers is 8.5mm. For the ballast, the properties of distributed discrete spring-damper elements are calculated from the provided information.

2.5. Simulation setup

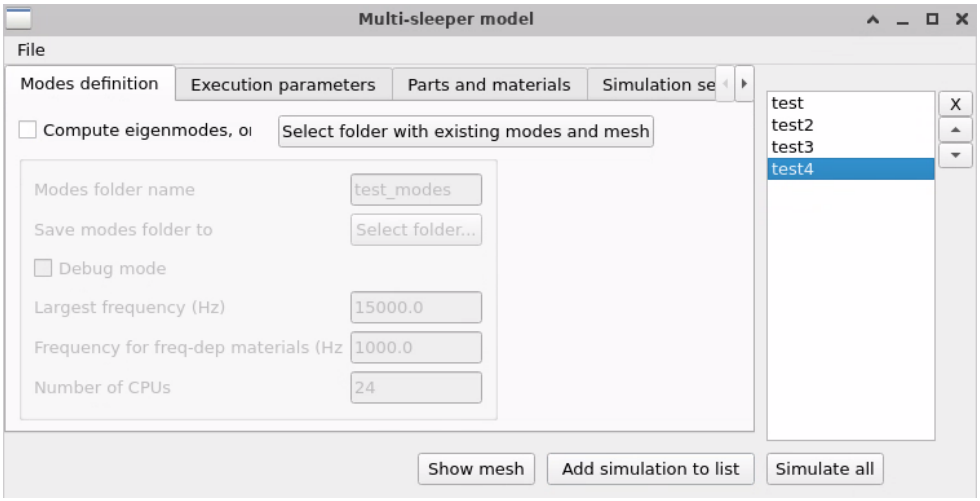
Fill the parameters according to the tool tips explanations. Note that:

- The excitation value is not a parameter because all outputs are normalized. However, its direction can be chosen. Positive lateral and vertical components mean outward of the track, and towards the ground respectively.
- Excitation application and FRFs extraction are by default always on the same side of the track, *i.e.* the right side when looking towards the X+ direction.
- Sleeper (or substructures, or *unit-cells*, etc.) numbering starts at 1.
- FRFs are extracted at all provided node groups of all selected substructures. If a group contains multiple nodes, the output is the node-wise average of the FRFs. Moreover, if the group name ends with "Y" or "Z", the output is the FRF in the vertical or the lateral direction, respectively. Otherwise, the magnitude is calculated, based on both directions.
- The sleeper range is used provided is used in the acoustic calculation as well as for MED result files
- For the acoustic part: if "2D" is requested, the acoustic mesh must be 2D and the FRF of acoustic power radiated through the surface is computed. If "1D", the acoustic pressure is calculated at all groups of nodes that only contain one node. If there are no such groups, the pressure is calculated at all nodes, arbitrary named "N1", N2", …

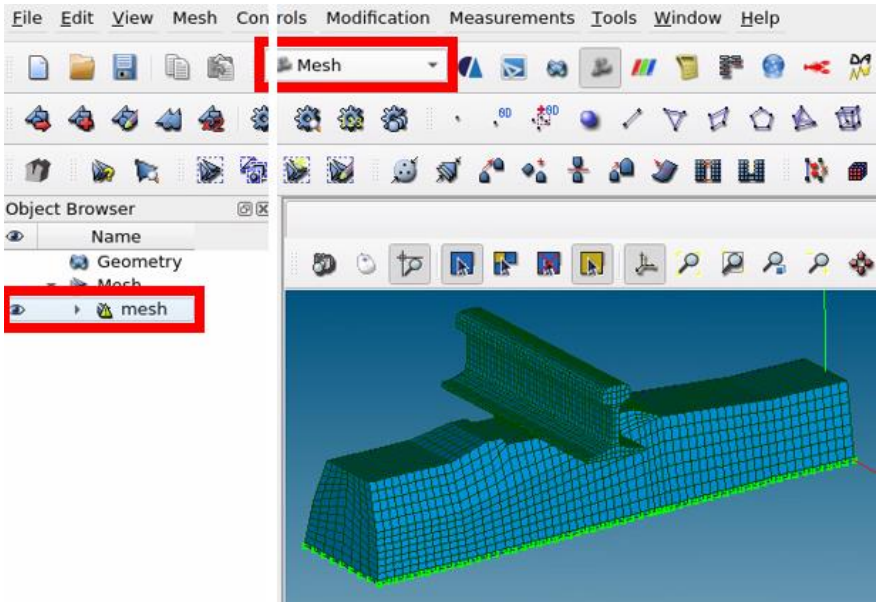


2.6. Create, manage and run simulations

Finally, click on “Add simulation to list”. Other simulations can be defined and appear in the list on the right. Selecting one of them displays all parameters that define it on the left of the GUI. In this example, “test4” will reuse modes that are computed beforehand: in this case, the ones of “test”.



The mesh of the macroelement can also be displayed with the button “Show mesh”. After Salome opens, go to the mesh module and show the mesh with the eye icon. Close Salome without saving to reuse the GUI.



Click on “Simulate all” to execute all saved simulations one by one. The terminal provides information to track simulations progress.

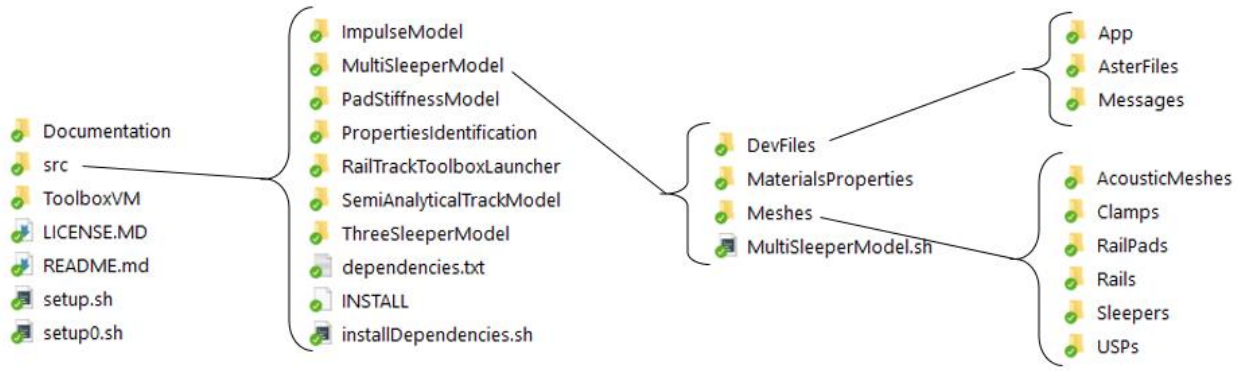
2.7. Results

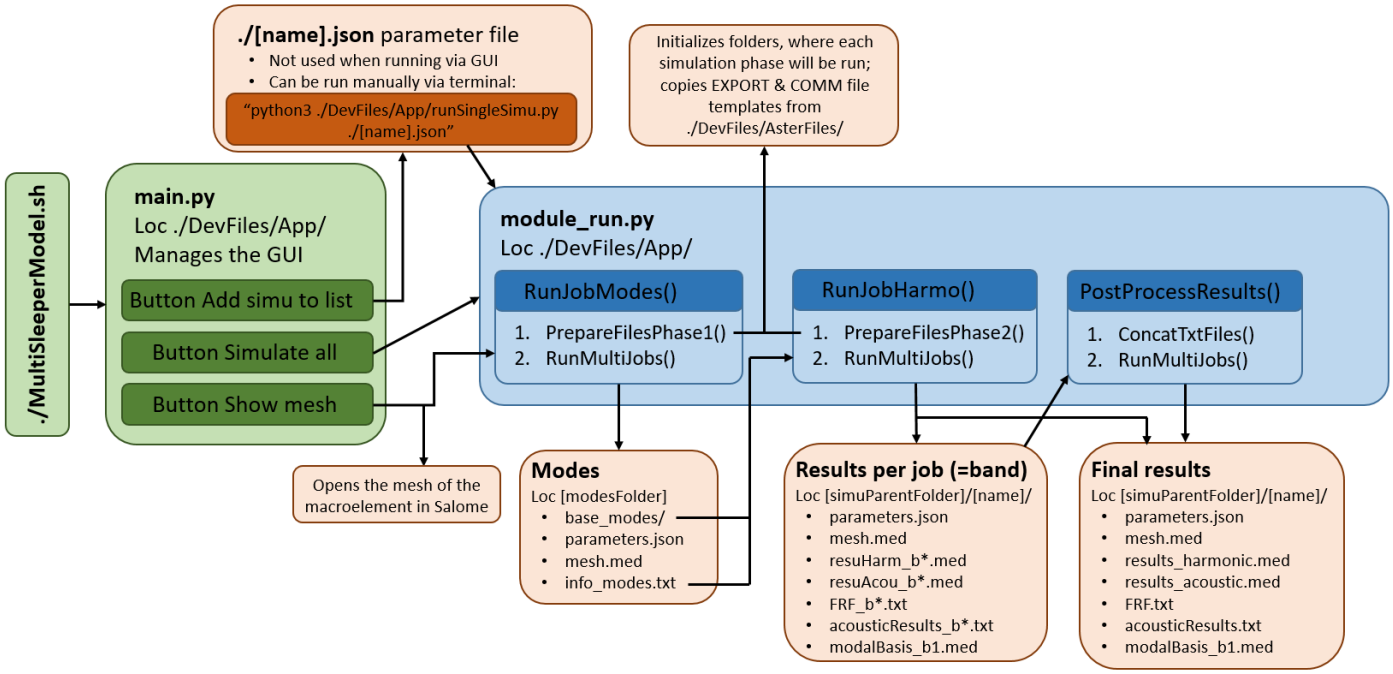
The results are in the “Output” folder of the requested directory:

- **results_harmonic.med**: file for visualization; it can be imported in the Paravis module of Salome to visualize the deformation of the track frequency by frequency, in the range of sleepers provided in *Simulation setup*.
- **results_acoustic.med**: same as above for the visualization of the acoustic field on the acoustic mesh. The field is analogous to a stress field and has the components 'SIXX' (real total pressure), 'SIYY' (imaginary total pressure), 'SIZZ' (real pressure from rails), 'SIXY' (imaginary pressure from rails), 'SIXZ' (real pressure from sleepers) and 'SIYZ' (imaginary pressure from sleepers).
- **FRF.txt**: text file containing the FRFs (magnitude and phase) at all requested groups of nodes of all selected substructures, column by column. The naming convention (title of the column) is "E[*sleeperNumber*]_[*groupName*]".
- **acousticResults.txt**: text file containing the acoustic FRFs. If "2D" was selected, the sound power level (SPL) is displayed at the top of the file, followed by the acoustic power FRF. If "1D" was selected, the acoustic pressure FRFs are displayed column by column for all the groups of nodes requested. See Developer Guide, 3.6 Acoustic calculation for more details.

3. Developer Guide

3.1. Code architecture flowchart





3.2. Code description

3.2.1. Generalities

The Multi-Sleeper is located in ./src/MultiSleeperModel/; all paths mentioned from now will be relative to that folder. The model GUI is launched with the script ./MultiSleeperModel.sh. It executes ./DevFiles/App/main.py, which opens and manages the GUI. The main.py allows collecting all inputs that the user provides via the GUI to generate a dictionary of parameters (see detailed description in X). The button "Add simu to list" appends it to the list of simulations self.simuList and exports it to the file ./[name].json, *name* being a key of the dictionnary and [name] the corresponding value: the name given to the simulation.

This JSON file is not used by the code, but it can be useful for the user to modify it manually. Moreover, it can be run manually without the GUI in a terminal by executing "python3 ./DevFiles/App/runSingleSimu.py

./[name].json". The main.py and runSingleSimu.py scripts identically use functions from ./DevFiles/App/module_run.py to execute simulations. The principal one is RunSimulation(), which calls other functions to run the first, the second and the post-processing phases successively.

In the end, in the kernel of the Multi-Sleeper Model, Code_Aster simulations are run. Most scripts of this model are used to correctly setup the simulations, manage the files exchanges and process the outputs. Any Code_Aster simulation ("job") is executed with a .export file and a .comm file. The EXPORT file contains several parameters such as RAM limits and number of CPUs, as well as a list of input/output files that are used in the job. The COMM (command) file is a python-based script with Code_Aster-specific functions, which contains the commands of what to do during that simulation. EXPORT and COMM files templates are located in ./DevFiles/AsterFiles/. See Code_Aster documentation D1.02.05 for EXPORT files syntax and https://code-aster.org/doc/v15/fr/index.php?man=commande for all Code_Aster functions documentation.

3.2.2. Simulation phase 1

The first phase consists in computing the eigenmodes of the macroelement. The function module_run.RunJobModes() firstly calls module_run.PrepareFilesPhase1(), which does the following:

1. Initialize the folder where the 1st phase is run. The directory is the value of *phase1WorkingDir* in the parameter dictionary, which is set by default to "[simuParentFolder]/[name]_phase1".
2. Copy the meshes and CSV materials property files
3. Copy computeModes1.export and computeModes.comm from the templates in ./DevFiles/AsterFiles/
4. Update computeModes1.export by finding and replacing string patterns

Then, the function module_run.RunMultiJobs() is used to execute the script ./DevFiles/App/runAsterJobs.sh, which allows executing Code_Aster simulations correctly with the EXPORT and the COMM files. During the execution, the COMM file does the following operations:

1. Assembly of components meshes
2. Definition and affectation of materials; note that materials properties are evaluated at the frequency defined by the dictionnary key *phase1Freq*.
3. Definition of boundary conditions, calculation of stiffness and mass matrices
4. Definition of frequency bands with INFO_MODE(): when calculating the modes, the spectrum from 0Hz to [phase1FreqMax]Hz needs to be divided in bands containing approximately 40 modes (see doc U4.52.02). INFO_MODE() is provided guessed bands and outputs the number of modes per band. Based on this output and under the assumption that modes are distributed homogeneously within each band, the new bands *freqBandsOpt* are defined with the algorithm implemented in OptimizedFreqBands(), such that each of them contains approximately 40 modes.
5. Modal simulation with CALC_MODES(); **warning**: if the first-guess bands (see previous point) too wide, for instance, the number of modes per band may differ enough to make the simulation crash.
6. Export modes to TXT and MED files
7. Suppression of all Code_Aster concepts but the modes *mdPh1* and the mesh DoF numbering *num1* and creation the "base" that which contains these concepts, *base_modes*, and which is meant to be reused in phase 2.

Finally, the files that are necessary in Phase 2 are copied to the directory [modesFolder]:

- base_modes/: the folder with the Code_Aster base that contains the eigenmodes
- parameters.json: a copy of the simulation parameters to keep track of what the parameters were, to compute these eigenmodes
- mesh.med: to visualize the macroelement mesh in Salome
- info_modes.txt: a table with modes informations; the first column is the mode number, the second one is the frequency, and the three last ones are the effective unit mass of the mode in the X, Y and Z directions respectively.

3.2.3. Simulation phase 2

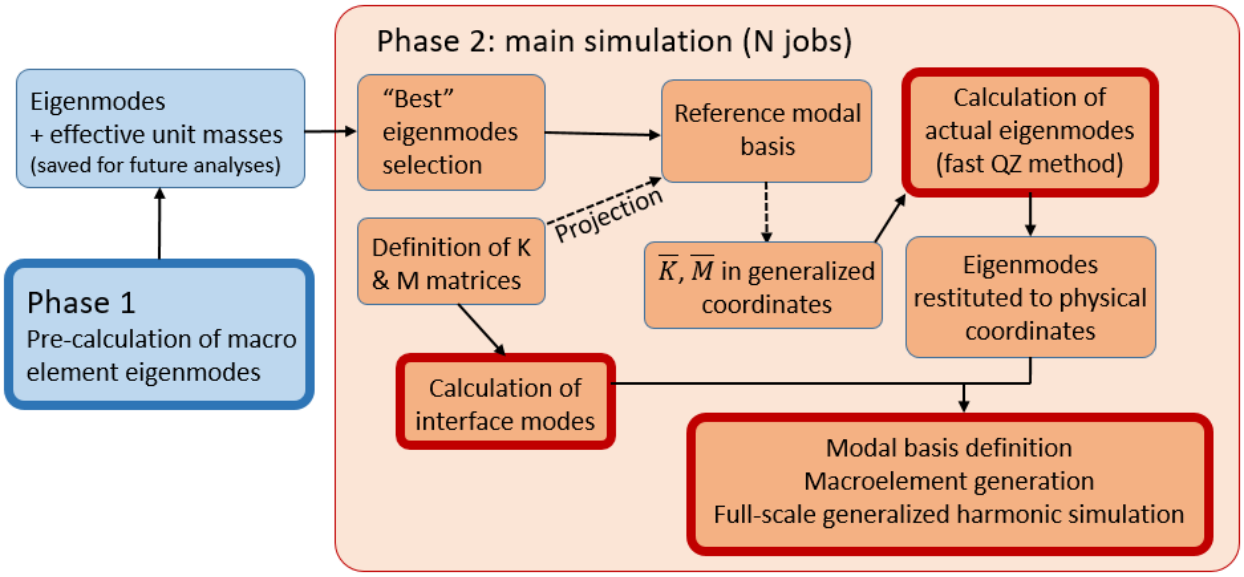
The second phase is the main harmonic simulation, divided into [nJobs] jobs. The function module_run.RunJobHarmo() firstly calls module_run.PrepareFilesPhase2(), which does the following:

1. Initialize the folder where the 2nd phase is run: [simuParentFolder]/[name]
2. Copy the necessary files from phase 1 (see above)
3. Copy the meshes and CSV materials property files
4. Use module_run.PrepareFreqFiles() to distribute the frequency list into [nJobs] input files “f1.txt” to “f[nJobs].txt” with lengths as constant as possible.
5. Copy runSimulation_b*.export and runSimulation.comm from the templates in ./DevFiles/AsterFiles/. There is one export file per job.
6. Update the runSimulation_b*.export files by finding and replacing string patterns

Then, the function module_run.RunMultiJobs() is used to execute the script ./DevFiles/App/runAsterJobs.sh, which allows executing Code_Aster simulations correctly with the EXPORT and the COMM files. During the execution, the COMM file does the following operations:

1. Assembly of components meshes
2. Definition and affectation of materials; note that in each job, materials storage moduli are evaluated at the frequency in the middle of the frequency range of the job. As for damping, it is optimized as explained in 3.5 Materials modelling.
3. The modal basis, which was computed in phase 1, may be very large and must be reduced to be used efficiently. The function GetEigenModesList() uses the information in info_modes.txt to select the N_x, N_y, N_Z modes with the largest effective unit mass along X, Y and Z respectively, such that the cumulated effective unit mass in each direction reaches [cumulMassEffeUn] (see Code_Aster documentation R05.01.03).
4. Calculation of stiffness mass and damping matrices
5. The stiffness and mass matrices are projected onto the reduced modal basis made of the mode shapes identified in 3. The mode shapes are the union of the sets of modes selected for each direction.
6. The modes of the macroelement are computed in generalized coordinates with the fast QZ method and restituted to physical coordinates with REST_GENE_PHYS() to obtain *mdPh2*, the N eigenmodes of the macroelement, which are sufficient to well describe its dynamics if [cumulMassEffeUn] is large enough (at least 0.94 to be accurate up to 2kHz).
   Note: this trick works under the hypothesis that the eigen shapes of the macroelement computed in phase 1 and those of the ones computed in phase 2 are similar. It is usually valid when materials properties do not vary drastically.

7. Interface modes are computed and added to the modal basis *mdPh2*
8. The macroelement *mcroEl1* is created and instantiated 2*[nSlp] times to build the generalized model (DEFI_MODELE_GENE())
9. The harmonic computation (DYNA_VIBRA()) is done in generalized coordinates



10. At every frequency, the velocity field is retrieved in physical coordinates; for each node of the acoustic mesh, the function computeDistance() creates a field on the radiant surfaces of the mesh, whose values are the distance to the acoustic mesh node. This field of distances is reused in computePressure() to evaluate the contribution of each radiant surface element to the complex acoustic pressure at the point of interest. The contributions are summed by integrating the radiant surfaces (POST_ELEM()). See 3.6 Acoustic calculation for more information.
    Note 1: in Code_Aster, fields of functions are used to calculate the real and imaginary acoustic pressure. The functions are for instance *pressFR[1,2,3]* and *pressFI[1,2,3]* and they are stored in the function fields *chFpr[1,2,3]* as two different components (namely X5, X6). To evaluate them once they are stored in a field, one needs to pass other fields as arguments. Among the fields passed, the following components are used in the formulas:
    - X1: component of the field of distances *chDist*; it simply denotes the distance to the acoustic mesh node.
    - X2: component of the field *CHk*, a homogeneous field, whose value is the wave number $k = \omega/c$.
    - X3: component of the field *CHcoef*, a homogeneous field, whose value is the coefficient $coef = \rho ck/2\pi$.
    - X4: component of the field *CH1* or *CHm1*, which are homogeneous fields with the value of 1 or -1 respectively. For substructures on the left of the track, which have been rotated of 180° about Y from the macroelement, a coefficient of -1 must be applied to the X and Z velocity components. Indeed, with REST_GENE_PHYS(), the velocity fields are expressed in the reference of the non-rotated macroelement.
    - DX, DY or DZ: depending on the direction of the contribution, this is the component of the real or imaginary velocity field *velR* or *velI*.

Note 2: before the calculations start, the acoustic mesh is analyzed. If the acoustic power is requested (2D mesh and [acMeshDim]='2D'), the acoustic pressure is calculated at every node in order to compute the acoustic power afterwards. If the acoustic pressure is requested, (1D or 2D mesh and [acMeshDim]='1D'), the acoustic pressure is calculated at all groups of nodes that only contain one node. If there are no such groups, the pressure is calculated at all nodes, arbitrary named "N1", N2", …

11. Acoustic pressure is written to resuAcou_b*.med; the field is analogous to a stress field and has the components 'SIXX' (real total pressure), 'SIYY' (imaginary total pressure), 'SIZZ' (real pressure from rails), 'SIXY' (imaginary pressure from rails), 'SIXZ' (real pressure from sleepers) and 'SIYZ' (imaginary pressure from sleepers).
12. Displacement is written to resuHarm_b*.med for the unit cells number [nSplAcoust1] to [nSlpAcoust2].
13. Acoustic pressure or power FRFs are written to acousticResults_b*.txt
14. Displacement, velocity or acceleration FRFs are written to FRF_b*.txt. The FRFs are queried at all groups of nodes in the list [nodesFRFs], and on all unit cells in the list [selectedSubstFRF]. Both lists should contain at least one element.
    Note: if an FRF is requested on a group that contains multiple nodes, the output is the node-wise average of the FRFs. Moreover, if the group name ends with "Y" or "Z", the output is the FRF in the vertical or the lateral direction, respectively. Otherwise, the magnitude is calculated, based on both directions.
15. One job (the 1st one) also exports modalBasis_b1.med, the modal basis that was used to make the macroelement, as well as skeleton.med, a mesh of the model for the unit cells number [nSplAcoust1] to [nSlpAcoust2].

3.2.4. Post-processing

The last simulation step is the post-processing of the results. It mainly consists in assembling the results of all frequencies, generated by the [nJobs] jobs, into one file per result type (MED, TXT, …). This is done with module_run.PostProcessResults(), which firstly calls module_run.ConcatTxtFiles().

1. It assembles all FRF_b*.txt to generate FRF.txt, which contains all displacement, velocity or acceleration transfer functions at the groups of nodes in [nodesFRF] and at all unit cells in [selectedSubstFRF]
2. It assembles all acousticResults_b*.txt to generate acousticResults.txt, which contains either the magnitude of acoustic pressure at all requested acoustic mesh nodes, or the acoustic power generated through the 2D acoustic mesh. Moreover, at the top of the file is displayed the sound power level (SPL). See 3.6 Acoustic calculation for more details.

Then, like in the previous phases, the templates postPro_concatMedFiles1.export and postPro_concatMedFiles.comm are copied to [simuParentFolder]/[name] and are executed using module_run.RunMultiJobs(). This job takes in input all resuHarm_b*.med and resuAcou_b*.med and assembles them with LIRE_RESU(), CREA_CHAMP() and CREA_RESU() to generate results_harmonic.med and results_acoustic.med.

3.3. Parameter file description

As explained previously, defining a simulation (button “Add simulation to list”) creates a python dictionary of parameters, which is passed throughout python functions as argument. The dictionary is written to the file ./[name].json. Here below are detailed all keys of this dictionary, which entirely defines a simulation.

Note: all keys of the dictionary are always defined, whatever the parameters are. For example, if USPs are not activated, the key *USPmesh* does exist but is set to *null*. Thus, if the user wants to modify the file such that USPs are included, there is no particular key to add; only values to edit.

E1Sleeper: storage modulus of the sleeper along the Z direction (float)
E2Sleeper: storage modulus of the sleeper along the Y direction (float)
E3Sleeper: storage modulus of the sleeper along the X direction (float)
ERail: storage modulus of the rail (float)
EUSP: path of the file with the frequency-dependent storage modulus of the USP (string)
Ebal: path of the file with the frequency-dependent storage modulus of the ballast (string)
Emat1: path of the file with the frequency-dependent storage modulus of the pad material 1 (string)
Emat2: path of the file with the frequency-dependent storage modulus of the pad material 2 (string)
USPMesh: path of the USP mesh file (string)
USP_on: activation of the USP (bool)
acMeshDim: dimension of the acoustic mesh (“1D” or “2D”, see 3.2.3 and 3.6)
acousticMesh: path of the acoustic mesh (string)
appPath: path of the app files (./DevFiles/App/)
balAreaCoef: effective loaded area in the ballast divided by the area of the bottom of the sleeper
clampDampX: clamp stiffness in the X direction
clampDampY: clamp stiffness in the Y direction
clampDampZ: clamp stiffness in the Z direction
clampStiffX: clamp damping in the X direction
clampStiffY: clamp damping in the Y direction
clampStiffZ: clamp damping in the Z direction
computeAcoustic: activation of the acoustic computation (bool)
computeModes: activation of the Phase 1 of the simulation (bool); if false, only Phase 2 is done
cumulMassEffeUn: cumulated effective unit mass (float); see 3.2.2, point 3.
cwd: path of the current working directory (./)
debugPh1: debug mode for phase 1 (bool); if true, the terminal does not close when job is over
debugPh2: debug mode for phase 2 (bool); if true, the terminals do not close when job is over
fDirLat: lateral component of the excitation vector direction (float); positive is outward of the track
fDirVert: vertical component of the excitation vector direction (float); positive is towards the ground
forceNode: name of the group of one node, where the excitation is applied (string)
frequencies: frequencies at which the second phase performs a harmonic simulation (list of float)
hBal: virtual height of the ballast in [m] (float); allows calculating the ballast discrete elements properties
host: name of the host server (string); by default: “localhost”
memLimit: memory limit [MB] for RAM usage in each job (float)
modesFolder: path of the folder where modes information is saved after phase 1, and/or recovered for phase 2 (string)
modesMaxFreq: largest frequency for modes calculation in phase 1 (float)

nCPUs: number of CPUs per job for phase 2 (int)
nJobs: number of jobs dividing the spectrum for phase 2 (int)
nModesRai: number of rail interface modes (int)
nModesSlp: number of sleeper interface modes (int)
nSlp: number of sleepers (int)
nSlpAcoust1: first sleeper, which contributes to the acoustic calculation and whose displacement field is printed to MED (int); sleeper numbering starts at 1
nSlpAcoust2: last sleeper, which contributes to the acoustic calculation and whose displacement field is printed to MED (int); sleeper numbering starts at 1
name: name of the simulation (string); all files are stored in [simuParentFolder]/[name]
nodesFRF: groups of nodes where FRFs are exported (list of string); see 3.2.4 point 14.
nuBal: Poisson's ratio of the ballast (float)
nuMat1: Poisson's ratio of the pad material 1 (float)
nuMat2: Poisson's ratio of the pad material 2 (float)
nuRail: Poisson's ratio of the rail (float)
nuSleeper: Poisson's ratio of the sleeper (float)
nuUSP: Poisson's ratio of the USP (float)
outputType: type of FRF exported (string: "DEPL", "VITE" OR "ACCE")
padMesh: path of the mesh of the rail pad (string)
phase1CPUs: number of CPUs used in phase 1 (int)
phase1Freq: frequency at which materials properties are evaluated in phase 1 (float)
phase1WorkingDir: path of the folder where the phase 1 is executed (string; by default: "[simuParentFolder]/[name]_phase1")
railMesh: path of the mesh of the rail (string)
reptrav: path of the folder where Code_Aster jobs are actually executed (string; by default: "/tmp/")
rhoRail: density of the rail in [kg/m3] (float)
rhoSleeper: density of the sleeper in [kg/m3] (float)
selectedSubstFRF: numbers of the unit cells, where FRFs are exported (list of int); numbering starts at 1
simuParentFolder: path of the folder where the folder [name] is created and results are stored (string)
sleeperMesh: path of the mesh of the sleeper (string)
slpForce: number of the unit cell, where the force is applied on mesh group [forceNode] (int; numbering starts at 1)
slpHeight: distance between the bottom of the sleeper and the middle of the face, which is in contact with the rail pad (float; value for B91 sleeper: 214mm; value for RpIV sleeper: 150mm)
slpSpacing: sleeper spacing (float)
tanDRail: damping ratio of the rail (float)
tanDSleeper: damping ratio of the sleeper (float)
tanDUSP: path of the file with the frequency-dependent damping ratio of the USP (string)
tanDbal: path of the file with the frequency-dependent damping ratio of the ballast (string)
tanDmat1: path of the file with the frequency-dependent damping ratio of the pad material 1 (string)
tanDmat2: path of the file with the frequency-dependent damping ratio of the pad material 2 (string)
thkUSP: thickness of the USP (float)
writeMED: activation of the exportation of visualization MED files (bool)

3.4. Meshes

All meshes are MED files that can be created in SALOME. Some meshes are available in the Meshes folder, classified component-wise. By default, the X-axis is the direction of the track, the Y-axis is the vertical direction and the Z-axis is the lateral direction. For nomenclature, X+ is “forward” such that Z+ is the right side of the track. The selected meshes of components are assembled correctly in Code_Aster during both simulation phases. By default, the assembled macroelement represents the right side of track unit cell.

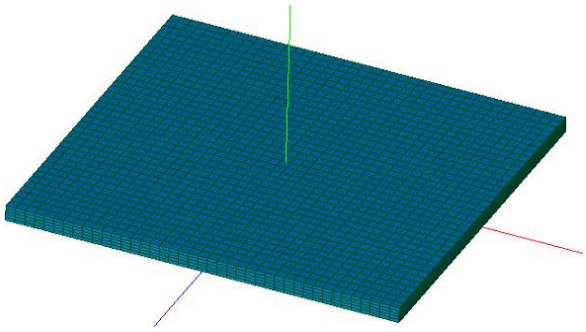
In addition to the mesh groups described below, there can be additional groups of nodes, where the frequency response functions (FRF) are extracted. Some are already present on the rail and the sleeper. Note that if the FRF is queried on a group of multiple nodes, the output is the average of the FRFs. Moreover, if the group name ends with “Y” or “Z”, the output is the FRF in the vertical or the lateral direction, respectively. Otherwise, the magnitude is calculated, based on the two directions.

3.4.1. Rail pad

It is by default 162mm x 7mm x 150mm, laying on the XZ plane and centered at the origin. The mandatory mesh groups are:

- “pnts”: group of nodes from the bottom fame (pad nodes to sleeper)
- “pntr”: group of nodes from the top face (pad nodes to rail)
- “hard”: group of volumes for the *Material 1* properties
- “soft”: group of volumes for the *Material 2* properties

The groups “hard” and “soft” were introduced to make bi-material pads for another project. The names were not changed but both materials can be anything. The only important thing is that both must exist and their union must contain all volume elements.

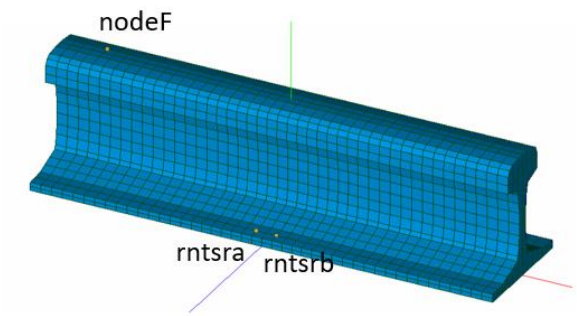


3.4.2. Rail

The mesh of the rail must be laying on the XZ plane as well and centered at the origin. The length of the portion of rail is 600mm by default. Note that when a different sleeper spacing is set, the code completely handles the transformation (scaling in X). The mandatory mesh groups are:

- “railBk”: nodes of the X- rail interface
- “railFt”: nodes of the X+ rail interface
- “rnts**”: 4 groups of one node (*rail node to sleeper*), “**” being either “ra”, “rb”, “la”, “lb”. It’s the nodes that are used to connect the clamp meshes. “r” for “right”, “l” for “left”, “a” for “back”, “b” for “front”

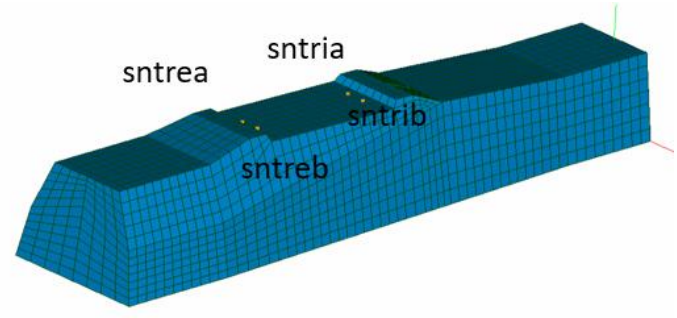
- A group of one node designating where the force is applied; by default, “nodeF” is proposed but the force can actually be applied anywhere.
- “railAcY”: group of radiant surfaces, which mainly face towards the Y-direction
- “railAcZ”: group of radiant surfaces, which mainly face towards the Z-direction
- “raile”: all volume elements of the mesh



3.4.3. Sleeper

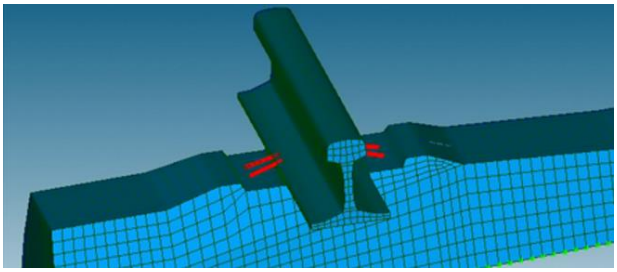
The mesh of the half sleeper is in the Z+ half space (right side of the track), laying on the XZ plane and centered at the origin. The mandatory mesh groups are:

- “slpSym”: group of nodes on the sleeper interface, where the sleeper is cut in half
- “sntb”: *sleeper nodes to ballast*; group of nodes of the bottom face, where the discrete elements modelling the ballast are placed
- “sntr**”: 4 groups of one node (*sleeper node to rail*), “**” being either “ia”, “ib”, “ea”, “eb”. It’s the nodes that are used to connect the clamp meshes. “i” for “interior”, “e” for “exterior”, “a” for “back”, “b” for “front”
- “slpAcX”: group of radiant surfaces, which mainly face towards the X-direction
- “slpAcY”: group of radiant surfaces, which mainly face towards the Y-direction
- “slpAcZ”: group of radiant surfaces, which mainly face towards the Z-direction
- “sftb”: *sleeper face to ballast*; group of face of the bottom of the sleeper. It is used to calculate the surface area, which must be considered to adjust the ballast discrete elements stiffness.
- “setp”: *sleeper elements to pad*; group of volume elements that are in contact with the pad. This group can actually contain all volume elements; it would just make the contacts detection longer.
- “setb”: *sleeper elements to ballast*; group of volume elements that are in contact with the ballast. Idem as above.
- “sleepere”: all sleeper volume elements



3.4.4. Clamp

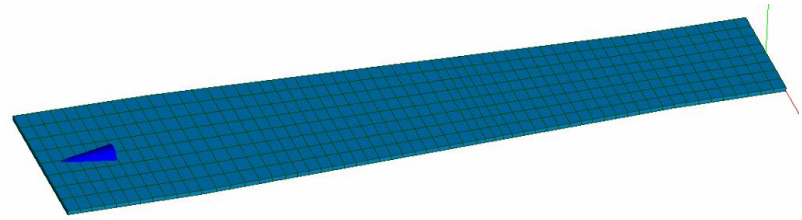
The clamp mesh cannot be chosen or modified. It simply bonds the nodes of the sleeper and of the rail described above using a discrete spring and damper element. It consists of a simple edge going from the origin to (1,0,0), and it is stretched, rotated and translated to be placed correctly and automatically on the macroelement.



3.4.5. USP

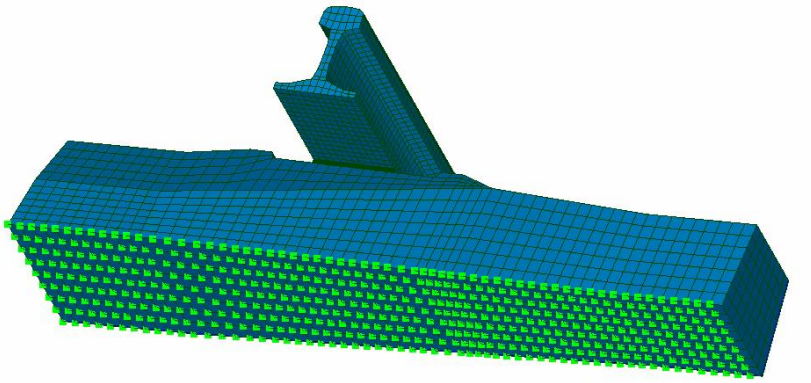
The mesh of the USP is placed exactly as described for the sleeper. Code_Aster manages automatically its placement with respect to the sleeper. To create one from a new sleeper, it is advised to make an extrusion of the bottom faces of the sleeper such that the geometry perfectly matches. Also, it is better to place only one liner element in the thickness, otherwise there will probably be too many hourglass-like modes to calculate.

The only mandatory group is “USPe” with all the volume elements of the USP.



3.4.6. Ballast

There is no mesh for the ballast. Its effective stiffness and damping are distributed on all nodes of the bottom of the sleeper, or of the bottom of the USP.



3.5. Materials modelling

To implement visco-elastic materials properties, the storage modulus E' and hysteretic damping ratio ($\tan\delta$) are provided for all materials except for the clamps, which are defined with viscous damping. The storage modulus and damping of all materials but the rail and the sleeper can be frequency-dependent. CSV files are used to describe these properties variation with frequency. The first column contains frequencies and the second one contains either the storage modulus in MPa, or the damping without units.

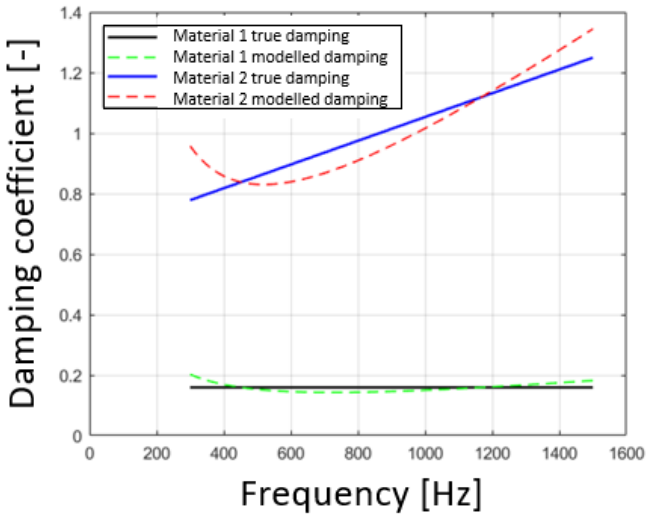
However, in this model, materials must be implemented with Rayleigh damping. The hysteretic damping is defined as

$$\eta = \frac{1}{2}\left(\alpha\omega + \frac{\beta}{\omega}\right)$$

where α, β are constants and $\omega = 2\pi f$ is the pulsation. In the 2nd phase of the simulation, both constants are optimized in the least square sense to model damping accurately:

$$\begin{bmatrix}\alpha_i \\ \beta_i\end{bmatrix} = \frac{2}{(\omega_{i+1}^3 - \omega_i^3)(\frac{1}{\omega_i} - \frac{1}{\omega_{i+1}}) - 3(\omega_{i+1} - \omega_i)^2}\begin{bmatrix} 3(\frac{1}{\omega_i} - \frac{1}{\omega_{i+1}}) & -3(\omega_{i+1} - \omega_i) \\ -3(\omega_{i+1} - \omega_i) & \omega_{i+1}^3 - \omega_i^3 \end{bmatrix}\begin{bmatrix}\int_{\omega_i}^{\omega_{i+1}} \eta(\omega)\cdot\omega \mathrm{d}\omega \\ \int_{\omega_i}^{\omega_{i+1}} \frac{\eta(\omega)}{\omega}\mathrm{d}\omega\end{bmatrix}$$

This equation is used in each job for all materials, where $[\omega_i, \omega_{i+1}]$ is the pulsation range of the i^{th} job.



For the ballast, sping-damper discrete elements are distributed on all nodes of the bottom of the sleeper or the USP. The total vertical and lateral stiffness are calculated as

$$k_{bal,Y} = \frac{E'_{bal}\cdot C\cdot A_{slp}}{h_{bal}}\ , \quad k_{bal,XZ} = \frac{k_{bal,Y}}{2(1+\nu_{bal})} \quad [\mathrm{N/mm}]$$

where $C \cdot A_{slp}$ is the effective loaded area in the ballast. And the total vertical and lateral damping

$$c_{bal,Y} = \frac{k_{bal,Y}\cdot\tan\delta}{\omega}\ , \quad c_{bal,XZ} = \frac{k_{bal,XZ}\cdot\tan\delta}{\omega} \quad [\mathrm{Ns/mm}]$$

all properties being constant within one frequency band.

Finally, the clamps properties provided by the user are the constant stiffness and viscous damping in the three directions.

3.6. Acoustic calculation

A monopole superposition technique is used to calculate the acoustic pressure at chosen locations. It consists in considering a set of radiant surfaces in the model as distributed monopole acoustic sources. The acoustic pressure at a distance r of a monopole source with surface area S and normal velocity of vibration $v_\perp$ is given by

$$p(r) = \frac{\rho c k v_\perp S}{4\pi r}$$

where ρ, c, k are respectively the air density, the speed of sound and the wave number. This formula is integrated over the radiant surfaces for each acoustic grid point. If the acoustic mesh is 2D and the user wants to calculate the acoustic power radiated through the surface, the acoustic intensity

$$I(\vec{x}, \omega) = \frac{p^2(\vec{x}, \omega)}{\rho c}$$

is integrated over the surface of the mesh. An important assumption is that all particles velocity vectors are perpendicular to the pressure waves isosurfaces and to the surface, on which the intensity is integrated. In other words, far-field quantities are computed and pressure waves are spherical.

Finally, from the acoustic power FRF P_{ac} in [W/N^2], the sound power level (SPL) is integrated from the acoustic power FRF:

$$L_w = 10 \cdot \log_{10}\left(\int_{f_{min}}^{f_{max}} P_{ac} \cdot \mathrm{d}f\right) \quad \left[\mathrm{dB}\left(\frac{\mathrm{W}}{\mathrm{N}^2}\right)\right].$$

3.7. Multiple macroelements for spatial variability

3.7.1. Description

Spatial variability was recently implemented in the context of the ToP-Noise project. This new functionality implies creating macroelements with different properties from the nominal one, and defining a custom sequence to place them. The optional second macroelement must be based on the same components meshes as the first macroelement. It can differ in terms of:

1. **Materials properties:** all materials properties except the ones of the rails can be different.
2. **Sleeper shift**: a sleeper shift can be introduced to model sleeper spacing variability. The shift is in the X-direction, *i.e.* the track main direction.

Note 1: if a non-zero sleeper shift is introduced in the second macroelement, a third macroelement with the negative value of that sleeper shift has to be generated as well to build the left side of the track. Indeed, this side is built with substructures that are rotated 180° about the vertical axis. Rotated substructures with a negative sleeper shift end up aligned with non-rotated (right side) substructures with a positive sleeper shift. Since that third macroelement is automatically created, it is therefore also available in the macroelement sequence definition.

Note 2: as explained in 3.2, the modes of the macroelements are computed in the first phase. It is the case as well for the optional 2nd and 3rd macroelements, which can make the first phase 2 to 3 times longer. If there is no sleeper shift in the second macroelement, its mesh is identical to the mesh of the main macroelement. Hence, one can choose to use the mode shapes of the main macroelement to efficiently compute the modes of the second macroelement in phase 2, the same way as it is done for the main macroelement (see 3.2.3. points 3. to 6.).

Note 3: with this new functionality, there are 5 possible sleeper spacings if a sleeper shift is introduced and depending on how the macroelements 1 (nominal), 2 and 3 are places:

1. Nominal sleeper spacing (configurations 1-1, 2-2, 3-3)
2. Nominal + sleeper shift (configurations 1-2, 3-1)
3. Nominal - sleeper shift (configurations 2-1, 1-3)
4. Nominal + 2x sleeper shift (configuration 3-2)
5. Nominal - 2x sleeper shift (configuration 2-3)

3.7.2. Use

This new functionality can be activated by editing a JSON parameter file, which was defined beforehand using the GUI, for instance. All dictionary keys, which are related to materials properties, define the properties of nominal macroelement (see 3.3). To introduce other macroelements, the two keys to manually add are:

- **macroEl2**: dictionary of materials properties that are assigned another value than in the main macroelement (dictionary; details below)
- **macroElSequence**: sequence of [nSlp] numbers indicating how macroelements are placed (list of int; values: 1 or 2 if no sleeper shift; 1, 2 or 3 if non-zero sleeper shift)

The value of the key *macroEl2* is a dictionary, which can contain the following keys:

- **slpShift**: sleeper shift in [mm] (float); optional keyword.
- **computeModeShapesPh1**: if there is no sleeper shift, allows to choose whether the mode shapes of the main macroelement are reused to compute the eigenmodes of the 2nd macroelement, or if the modes of the 2nd macroelement must be computed in phase 1 as well (see 3.7.1, note 2).

Finally, all materials properties that are inserted in [macroEl2] as new keys are used to create the second and the eventual third macroelement. All the ones that are not explicitly informed are assumed to be identical to the nominal macroelement. Any of the following properties can be inserted and assigned the same type of value as for the main macroelement (see 3.3):

- **Sleeper**: *E1Sleeper, E2Sleeper, E3Sleeper, nuSleeper, tanDSleeper, rhoSleeper*
- **USP**: *EUSP, nuUSP, tanDUSP*
- **Pad materials 1 and 2**: *Emat1, Emat2, nuMat1, nuMat2, tanDmat1, tanDmat2*
- **Ballast**: *Ebal, nuBal, tanDbal*
- **Clamp**: *clampStiffX, clampStiffY, clampStiffZ, clampDampX, clampDampY, clampDampZ*

3.7.3. Model modifications

Below are listed the main modifications in the code of the model. Firstly, all actions related to the implementation of a new macroelement are based on the three following boolean variables, which are defined in many functions or scripts:

1. **bool_macroEl2_Ph1**: true if the modes of the second macroelement must be computed specifically in Phase 1. It is automatically the case if there is a non-zero sleeper shift ([macroEl2][slpShift]!=0), or simply if the user requests it by setting [macroEl2][computeModeShapesPh1] to True.
2. **bool_macroEl2_Ph2**: true if a second macroelement is going to be defined and used in phase 2, whether its modes are pre-computed in phase 1 or not.
3. **bool_macroEl2_slpShift**: true if there is a sleeper shift in the 2nd macroelement, such that a 3rd element with minus the sleeper shift value needs to be created as well.

The most notable modifications done to the code can be summarized as follows:

- For all materials properties that are different in the second macroelement and related to a file (frequency-dependent properties), the concerned files are also copied to the working directory with the suffix "_2". Ex. storage modulus of the ballast of the 2nd (and 3rd) macroelement(s): "Ebal_2.csv".
- Many Code_Aster concepts have to be created a 2nd (or 3rd) time specifically for the other macroelements, such as the mesh, the model and all concepts that are based on them, such as fields in phase 2. The ones related to the second and third macroelement have the suffix "2" and "2S", respectively, "S" for "symmetrical".
- Out of phase 1 is also exported the modal information for the 2nd macroelement info_modes2.txt, which is the same as the one for the 3rd macroelement since they are symmetrical. Moreover, the base reused in phase 2 also contains the concepts *mdPh1_2* and *num1_2* as well as *mdPh1_2S* and *num1_2S* if there is a sleeper shift, hence a 3rd macroelement.

3.8. Future work

In the first phase, there might be no need to redo a STURM verification of the modes in CALC_MODES(). It is already what INFO_MODE() does and it takes quite a long time. So there is possibly something to look at in the key word VERI_MODE of the function CALC_MODES(), in order to disable this verification step.

Still in the first phase, the modal simulations (INFO_MODES() + CALC_MODES()) are done one after the other when other macroelements are requested. It would be more efficient in parallel, such that the eigenmodes of each macroelement are computed on separate jobs. In this case, the problem is that it would generate multiple Code_Aster bases to be reused in the 2nd phase, however it is only possible to use one. The trick could be to export the modes in MED files, that would be reimported in phase 2 as a MODE_MECA concept with LIRE_RESU(). There are examples in the validation documentation for the function MAC_MODES() (see U4.52.15).