pymls Documentation

Release 1.4.6

O. Dazel, M. Gaborit, P. Göransson

May 29, 2020

Contents:

1 Materials **3**

2 API documentation **5**
- 2.1 pymls.solver module . . . 5
- 2.2 pymls.backing module . . . 7
- 2.3 pymls.analysis module . . . 7
- 2.4 pymls.interface package . . . 7
- 2.5 pymls.layers package . . . 8
- 2.6 pymls.media package . . . 9
- 2.7 pymls.utils package . . . 9
- 2.8 Module contents . . . 10

3 Indices and tables **11**

Python Module Index **13**

Index **15**

pymls allows to solve acoustic propagation problems through structures with multiple layers using Dazel *et al.*'s method as described in "A stable to model the acoustic response of multilayered structures" (Journal of Applied Physics, 2013, doi: 10.1063/1.4790629).

CHAPTER 1

Materials

Materials are described using classes from `pymls.media` module. They can be either created from scratch by instanciating the corresponding class and inserting values into the parameters or reading YAML files.

CHAPTER 2

API documentation

2.1 pymls.solver module

exception pymls.solver.**IncompleteDefinitionError**(*msg='The definition is incomplete and no analysis can be performed'*)

Bases: Exception

Exception raised when attempting to solve an incomplete system

Either missing layers definition or no backing will produce such an error.

class pymls.solver.**Solver**(*media=None*, *analyses=None*, *layers=None*, *backing=None*)

Bases: object

Stores a system to solve and parameters for the analysis.

Performs analysis and gives back raw unmodified/cleaned results. All post-processing should be done *out* of this class.

Parameters

- **layers** (*list of Layer/StochasticLayer instances*) – The right most layer appears last in the list.
- **backing** (function reference from *pymls.backing*) – Describe the type of backing condition
- **media** (*list of Media subclasses, optional*) – Stores all media used in the system for later reference
- **analyses** (*list of Analysis instances, optional*) – If only one instance is provided with a list, the constructor will wrap it into a list.

media
list of Media subclasses, optional

layers
list of Layer/StochasticLayer instances

backing
function reference from *pymls.backing*

analyses
list of Analysis instances, optional

resultset
list of dict – Contains the results for all analysis and metadata

solve(frequencies, angles, n_draws, prng_state) : list of dict
Starts the solving process w/w stochastic parameters.

check_is_complete() : bool
Check that all required data has been provided and gathers media.

Methods

check_is_complete()	Check that all required data has been provided and gathers media.
compute_fields(layer_id, frequency, theta_inc)	Returns the backpropagation matrix from the first interface to layer num.
solve([frequencies, angles, n_draws, prng_state])	Starts the solving process w/w stochastic parameters.

check_is_complete()
Check that all required data has been provided and gathers media.

Returns True if the described is complete and ready to be solved.

Return type bool

Raises *IncompleteDefinitionError* – If the system is incomplete (missing layer or backing)

compute_fields(*layer_id*, *frequency*, *theta_inc*)
Returns the backpropagation matrix from the first interface to layer num. *layer_id*.

Parameters

- **frequency** (*float*) – frequency at which backprop is computed
- **theta_inc** – angle of incidence
- **layer_id** (*int*) – id of the layer up to which backpropagate

Returns **layer_func** – Function to get the propagation in the layer

Return type callable

Raises *ValueError :* – if the id is invalid

solve(*frequencies=None*, *angles=0*, *n_draws=1000*, *prng_state=None*)
Starts the solving process w/w stochastic parameters.

The function looks for *StochasticLayer* instances in the *layers* list and flag them. It creates an *Analysis* if all parameters are provided upon call and runs the corresponding solver functions for all registered analysis, gathering results in *resultset*.

Parameters

- **frequencies** (*list, optional*) – list of frequency where to compute the analysis. If it isn't provided and no *Analysis* has been registered before hand, the function will return nothing.
- **angles** (*optional*) – Defaults to 0. Can be a list or anything *Analysis* can parse to an iterable.
- **n_draws** (*int*) – Number of draws for the stochastic analyses.
- **prng_state** (*tuple*) – Saved state for Numpy's pseudo random number generator (see *numpy.random.get_state*)
- **_numpy.random.get_state** (*.*) –

Returns **resultset** – Set of all computed results and relevant metadata provided as a dict for easy serialisation.

Return type dict or list of dict

2.2 pymls.backing module

pymls.backing.**rigid**(*omega*, *k_x*)

pymls.backing.**transmission**(*omega*, *k_x*)

2.3 pymls.analysis module

class pymls.analysis.**Analysis**(*name*, *freqs*, *angles*, *enable_stochastic=False*)

Bases: object

2.4 pymls.interface package

2.4.1 Submodules

2.4.2 pymls.interface.interfaces module

pymls.interface.interfaces.**elastic_fluid_interface**(*O*)

pymls.interface.interfaces.**elastic_pem_interface**(*O*)

pymls.interface.interfaces.**fluid_elastic_interface**(*O*)

pymls.interface.interfaces.**fluid_pem_interface**(*O*)

pymls.interface.interfaces.**pem_elastic_interface**(*O*)

pymls.interface.interfaces.**pem_fluid_interface**(*O*)

2.4.3 pymls.interface.utils module

pymls.interface.utils.**generic_interface**(*medium_left*, *medium_right*)

Returns a callable to the interface function corresponding to the two given media.

Note: interface functions are not symmetrical (generic_interface(m1, m2) != generic_interface(m2, m1)).

pymls.interface.utils.**rigid_interface**(*medium*)
Returns a callable to the rigid backing function corresponding to the given media.

2.4.4 Module contents

2.5 pymls.layers package

2.5.1 Submodules

2.5.2 pymls.layers.elastic module

pymls.layers.elastic.**transfert_elastic**(*Omega_minus*, *omega*, *k_x*, *medium*, *d*)

2.5.3 pymls.layers.fluid module

pymls.layers.fluid.**transfert_fluid**(*Omega_minus*, *omega*, *k_x*, *medium*, *d*)

2.5.4 pymls.layers.layer module

class pymls.layers.layer.**Layer**(*medium*, *thickness*, *name='Unnamed Layer'*)
Bases: object

Methods

register	
update_frequency	

register(*hook_name*)

update_frequency(*omega*)

class pymls.layers.layer.**StochasticLayer**(*medium*, *thickness*, *stochastic_param*, *pdf*, *name='Unnamed Layer'*)
Bases: *pymls.layers.layer.Layer*

Methods

register	
reinit	
update_frequency	

reinit()

2.5.5 pymls.layers.pem module

pymls.layers.pem.**transfert_pem**(*Omega_minus*, *omega*, *k_x*, *medium*, *d*)

2.5.6 pymls.layers.screen module

pymls.layers.screen.**transfert_screen**(*Omega_minus*, *omega*, *k_x*, *m*, *d*)

2.5.7 pymls.layers.utils module

pymls.layers.utils.**generic_layer**(*medium*)

2.5.8 Module contents

2.6 pymls.media package

2.6.1 Module contents

pymls.media.medium

pymls.media.fluid

pymls.media.eqf

pymls.media.air

pymls.media.elastic

pymls.media.pem

pymls.media.screen

2.7 pymls.utils package

2.7.1 Submodules

2.7.2 pymls.utils.hdf5_export module

2.7.3 pymls.utils.indicators module

pymls.utils.indicators.**TL_from_T**(*T*)
Compute the Transmission Loss from the Transmission coefficient

pymls.utils.indicators.**alpha_from_R**(*R*)
Compute the absorption coefficient from the Reflexion coefficient

2.7.4 pymls.utils.yaml_loader module

class pymls.utils.yaml_loader.**YamlLoader**
Bases: object

Load a multilayer definition from a yaml file

Methods

extract_from_yaml	
from_file	
parse_yaml	
yaml_is_valid	

```
EXPECTED_FIELDS = {'analysis': {'type': [<class 'list'>, <class 'dict'>], 'item_keys
KEYS_ANALYSIS = {'frequency': ['type', 'value'], 'range': ['type', 'start', 'end', '
MAP_BACKING = {'rigid': <function rigid at 0x7f74b3e63510>, 'transmission': <functio
```

extract_from_yaml(*yaml=None*)

from_file(*filename*)

parse_yaml()

yaml_is_valid()

2.7.5 Module contents

2.8 Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

pymls, 10
pymls.analysis, 7
pymls.backing, 7
pymls.interface, 8
pymls.interface.interfaces, 7
pymls.interface.utils, 7
pymls.layers, 9
pymls.layers.elastic, 8
pymls.layers.fluid, 8
pymls.layers.layer, 8
pymls.layers.pem, 8
pymls.layers.screen, 9
pymls.layers.utils, 9
pymls.solver, 5
pymls.utils, 10
pymls.utils.indicators, 9
pymls.utils.yaml_loader, 9

Index

A

alpha_from_R() (in module pymls.utils.indicators), 9
analyses (pymls.solver.Solver attribute), 6
Analysis (class in pymls.analysis), 7

B

backing (pymls.solver.Solver attribute), 5

C

check_is_complete() (pymls.solver.Solver method), 6
compute_fields() (pymls.solver.Solver method), 6

E

elastic_fluid_interface() (in module pymls.interface.interfaces), 7
elastic_pem_interface() (in module pymls.interface.interfaces), 7
EXPECTED_FIELDS (pymls.utils.yaml_loader.YamlLoader attribute), 10
extract_from_yaml() (pymls.utils.yaml_loader.YamlLoader method), 10

F

fluid_elastic_interface() (in module pymls.interface.interfaces), 7
fluid_pem_interface() (in module pymls.interface.interfaces), 7
from_file() (pymls.utils.yaml_loader.YamlLoader method), 10

G

generic_interface() (in module pymls.interface.utils), 7
generic_layer() (in module pymls.layers.utils), 9

I

IncompleteDefinitionError, 5

K

KEYS_ANALYSIS (pymls.utils.yaml_loader.YamlLoader attribute), 10

L

Layer (class in pymls.layers.layer), 8
layers (pymls.solver.Solver attribute), 5

M

MAP_BACKING (pymls.utils.yaml_loader.YamlLoader attribute), 10
media (pymls.solver.Solver attribute), 5

P

parse_yaml() (pymls.utils.yaml_loader.YamlLoader method), 10
pem_elastic_interface() (in module pymls.interface.interfaces), 7
pem_fluid_interface() (in module pymls.interface.interfaces), 7
pymls (module), 10
pymls.analysis (module), 7
pymls.backing (module), 7
pymls.interface (module), 8
pymls.interface.interfaces (module), 7
pymls.interface.utils (module), 7
pymls.layers (module), 9
pymls.layers.elastic (module), 8
pymls.layers.fluid (module), 8
pymls.layers.layer (module), 8
pymls.layers.pem (module), 8
pymls.layers.screen (module), 9
pymls.layers.utils (module), 9
pymls.solver (module), 5
pymls.utils (module), 10
pymls.utils.indicators (module), 9
pymls.utils.yaml_loader (module), 9

R

register() (pymls.layers.layer.Layer method), 8
reinit() (pymls.layers.layer.StochasticLayer method), 8
resultset (pymls.solver.Solver attribute), 6
rigid() (in module pymls.backing), 7

rigid_interface() (in module pymls.interface.utils), 7

S

solve() (pymls.solver.Solver method), 6
Solver (class in pymls.solver), 5
StochasticLayer (class in pymls.layers.layer), 8

T

TL_from_T() (in module pymls.utils.indicators), 9
transfert_elastic() (in module pymls.layers.elastic), 8
transfert_fluid() (in module pymls.layers.fluid), 8
transfert_pem() (in module pymls.layers.pem), 8
transfert_screen() (in module pymls.layers.screen), 9
transmission() (in module pymls.backing), 7

U

update_frequency() (pymls.layers.layer.Layer method), 8

Y

yaml_is_valid() (pymls.utils.yaml_loader.YamlLoader method), 10
YamlLoader (class in pymls.utils.yaml_loader), 9