pelicun Documentation

Release 1.0.0

Adam Zsarnoczay

Nov 22, 2019

CONTENTS

pelicun

Probabilistic Estimation of Losses, Injuries, and Community resilience Under Natural disasters

pelicun is a Python package that provides tools for assessment of damage and losses due to natural hazards. It uses a stochastic damage and loss model that is based on the methodology described in FEMA P58 (FEMA, 2012). While FEMA P58 aims to assess the seismic performance of a building, with *pelicun* we want to develop a more versatile, hazard-agnostic tool that will eventually provide loss estimates for other types of assets (e.g. bridges, facilities, pipelines) and lifelines. The underlying loss model was designed with these objectives in mind and it will be gradually extended to have such functionality.

Currently, the scenario assessment from the FEMA P58 methodology is built-in the tool. Detailed documentation of the available methods and their use is available at http://pelicun.readthedocs.io

The current version of *pelicun* can be used to quantifiy lossess from an earthquake scenario in the form of *decision variables*. This functionality is typically utilized for performance based engineering or seismic risk assessment. There are several steps of seismic performance assessment that *pelcicun* can help with:

- **Describe the joint distribution of seismic response.** The response of a structure or other type of asset to an earthquake is typically described by so-called *engineering demand parameters* (EDPs). *pelicun* provides methods that take a finite number of EDP vectors and find a multivarite distribution that describes the joint distribution of EDP data well.
- **Define the damage and loss model of a building.** The component damage and loss data from FEMA P58 is provided with *pelicun*. This makes it easy to define building components without having to provide all the data manually. The stochastic damage and loss model is designed to facilitate modeling correlations between several parameters of the damage and loss model.
- **Estimate component damages.** Given a damage and loss model and the joint distribution of EDPs, *pelicun* provides methods to estimate the quantity of damaged components and collapses.
- **Estimate consequences.** Using information about collapses and component damages, the following consequences can be estimated with the loss model: reconstruction cost and time, unsafe placarding (red tag), injuries and fatalities.

CHAPTER

ONE

INSTALLATION

pelicun is available for Python 2.7 and Python 3.5+ at the Python Package Index (PyPI). You can simply install it using `pip` as follows:

```
pip install pelicun
```

1.1 Requirements

The following packages are required for *pelicun*:

package	minimum version
numpy	1.15.1
scipy	1.1
pandas	0.20

We recommend installing the Anaconda Python distribution because these packages and many other useful ones are available there.

CHAPTER

TWO

FEATURES

The following table outlines the features that are currently available in the tool and the requirements that will drive future development. We welcome suggestions for useful features that are missing from the list below. The priority column provides information about the relative importance of features planned for a given release: **M** - mandatory, **D** - desirable, **O** - optional, **P** - possible.

Table 1: List of features

ID	description	priority	available
1	**Assessment Methods**		
1.1	Perform component-based (e.g. FEMA-P58 style) loss assessment for earthquake scenarios.	M	1.0
1.2	Perform component-group-based (e.g HAZUS style) loss assessment for earthquake scenarios.	D	1.1
1.3	Perform loss assessment for hurricane scenarios based on the HAZUS hurricane methodology.	D	1.2
1.4	Perform downtime estimation using the ARUP's REDi methodology.	D	1.2
1.5	Perform time-based assessment for seismic hazard.	M	1.3
2	**Control**		
2.1	Specify number of realizations.	M	1.0
2.2	Specify log-standard deviation increase to consider additional sources of uncertainty.	M	1.0
2.3	Pick the decision variables to calculate.	D	1.0
2.4	Specify the number of inhabitants on each floor and their temporal distribution.	D	1.0
2.5	Specify the basic boundary conditions of repairability.	D	1.0
2.6	Control collapse through EDP limits.	D	1.0
2.7	Specify the replacement cost and time for the asset.	M	1.0
2.8	Specify EDP boundaries that define the domain with reliable simulation results.	D	1.0
2.9	Specify collapse modes and characterize the corresponding likelihood of injuries.	D	1.0
3	**Component DL information**		
3.1	Make the component damage and loss data from FEMA P58 (1st ed.) available in the tool.	M	1.0
3.2	Facilitate the use of custom components for loss assessment.	D	1.0
3.3	Enable different component quantities for each floor in each direction.	D	1.0
3.4	Enable fine control over quantities of identical groups of components within a PG.	D	1.0
3.5	Create a generic JSON data format to store component DL data.	D	1.1
3.6	Convert FEMA P58 and HAZUS component DL data to the new JSON format.	D	1.1
3.7	Extend the list of available decision variables with those from HAZUS	D	1.2
3.8	Extend the list of available decision variables with those from REDi	D	1.2
4	**Stochastic loss model**		
4.1	Enable control of basic dependencies between logically similar parts of the model.	D	1.0
4.2	Enable control of basic dependencies between reconstruction cost and reconstruction time.	D	1.0
4.3	Enable control of basic dependencies between different levels of injuries.	D	1.0
4.4	Extend the model to include the description of the hazard (earthquake and hurricane).	D	1.3
4.5	Enable finer control of dependencies through intermediate levels of correlation.	D	1.3

Continued on next page

Table 1 – continued from previous page

ID	description	priority	available	planned
5	**Response estimation**			
5.1	Fit a multivariate random distribution to samples of EDPs from response simulation.	M	1.0	
5.2	Allow estimation of EDPs using empirical functions instead of simulation results.	D		1.2
5.3	Perform EDP estimation using the empirical functions in the HAZUS earthquake method	D		1.2

2.1 Releases

Minor releases are planned to follow quarterly cycles while major releases are planned at the end of the third quarter every year:

Table 2: Release schedule

version	planned release date
1.0	Oct 2018
1.1	Dec 2018
1.2	March 2019
1.3	June 2019
2.0	Sept 2019

CHAPTER

THREE

COPYRIGHT AND LICENSE

The *pelicun* Python package is copyright through Leland Stanford Junior University and The Regents of the University of California.

The software is distributed under the BSD 3-Clause License.

pelicun leverages several third-party software packages, which have separate licensing policies.

3.1 Copyright

```
Copyright (c) 2018, Leland Stanford Junior University
Copyright (c) 2018, The Regents of the University of California
```

3.2 BSD 3-Caluse license

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

* Neither the name of the copyright holder nor the names of its
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

CHAPTER

FOUR

API DOCUMENTATION:

4.1 Modules

4.1.1 pelicun.base module

This module defines constants, basic classes and methods for pelicun.

4.1.2 pelicun.control module

This module has classes and methods that control the loss assessment.

Contents

Assessment()	A high-level class that collects features common to all supported loss assessment methods.
FEMA_P58_Assessment([inj_lvls])	An Assessment class that implements the loss assessment method in FEMA P58.

class pelicun.control.**Assessment**

Bases: object

A high-level class that collects features common to all supported loss assessment methods. This class will only rarely be called directly when using pelicun.

Attributes

beta_tot Calculate the total additional uncertainty for post processing.

Methods

calculate_damage(self)	Characterize the damage experienced in each random event realization.
calculate_losses(self)	Characterize the consequences of damage in each random event realization.
define_loss_model(self)	Create the stochastic loss model based on the inputs provided earlier.

Continued on next page

Table 2 – continued from previous page

define_random_variables(self)	Define the random variables used for loss assessment.
read_inputs(self, path_DL_input, path_EDP_input)	Read and process the input files to describe the loss assessment task.
write_outputs(self)	Export the results.

property beta_tot
Calculate the total additional uncertainty for post processing.

The total additional uncertainty is the squared root of sum of squared uncertainties corresponding to ground motion and modeling.

Returns

beta_total: float The total uncertainty (logarithmic EDP standard deviation) to add to the EDP distribution. Returns None if no additional uncertainty is assigned.

read_inputs(*self*, *path_DL_input*, *path_EDP_input*, *verbose=False*)
Read and process the input files to describe the loss assessment task.

Parameters

path_DL_input: string Location of the Damage and Loss input file. The file is expected to be a JSON with data stored in a standard format described in detail in the Input section of the documentation.

path_EDP_input: string Location of the EDP input file. The file is expected to follow the output formatting of Dakota. The Input section of the documentation provides more information about the expected formatting.

verbose: boolean, default: False If True, the method echoes the information read from the files. This can be useful to ensure that the information in the file is properly read by the method.

define_random_variables(*self*)
Define the random variables used for loss assessment.

define_loss_model(*self*)
Create the stochastic loss model based on the inputs provided earlier.

calculate_damage(*self*)
Characterize the damage experienced in each random event realization.

calculate_losses(*self*)
Characterize the consequences of damage in each random event realization.

write_outputs(*self*)
Export the results.

class pelicun.control.**FEMA_P58_Assessment**(*inj_lvls=2*)
Bases: *pelicun.control.Assessment*

An Assessment class that implements the loss assessment method in FEMA P58.

Attributes

beta_tot Calculate the total additional uncertainty for post processing.

Methods

aggregate_results(self)	
calculate_damage(self)	Characterize the damage experienced in each random event realization.
calculate_losses(self)	Characterize the consequences of damage in each random event realization.
define_loss_model(self)	Create the stochastic loss model based on the inputs provided earlier.
define_random_variables(self)	Define the random variables used for loss assessment.
read_inputs(self, path_DL_input, path_EDP_input)	Read and process the input files to describe the loss assessment task.
write_outputs(self)	

read_inputs (*self*, *path_DL_input*, *path_EDP_input*, *verbose=False*)

Read and process the input files to describe the loss assessment task.

Parameters

path_DL_input: string Location of the Damage and Loss input file. The file is expected to be a JSON with data stored in a standard format described in detail in the Input section of the documentation.

path_EDP_input: string Location of the EDP input file. The file is expected to follow the output formatting of Dakota. The Input section of the documentation provides more information about the expected formatting.

verbose: boolean, default: False If True, the method echoes the information read from the files. This can be useful to ensure that the information in the file is properly read by the method.

define_random_variables (*self*)

Define the random variables used for loss assessment.

Following the FEMA P58 methodology, the groups of parameters below are considered random. Simple correlation structures within each group can be specified through the DL input file. The random decision variables are only created and used later if those particular decision variables are requested in the input file.

1. Demand (EDP) distribution

Describe the uncertainty in the demands. Unlike other random variables, the EDPs are characterized by the EDP input data provided earlier. All EDPs are handled in one multivariate lognormal distribution. If more than one sample is provided, the distribution is fit to the EDP data. Otherwise, the provided data point is assumed to be the median value and the additional uncertainty prescribed describes the dispersion. See _create_RV_demands() for more details.

2. Component quantities

Describe the uncertainty in the quantity of components in each Performance Group. All Fragility Groups are handled in the same multivariate distribution. Consequently, correlation between various groups of component quantities can be specified. See _create_RV_quantities() for details.

3. Fragility EDP limits

Describe the uncertainty in the EDP limit that corresponds to exceedance of each Damage State. EDP limits are grouped by Fragility Groups. Consequently, correlation between fragility limits are currently limited within Fragility Groups. See _create_RV_fragilities() for details.

4. Reconstruction cost and time

Describe the uncertainty in the cost and duration of reconstruction of each component conditioned on the damage state of the component. All Fragility Groups are handled in the same multivariate distribution. Consequently, correlation between various groups of component reconstruction time and cost estimates can be specified. See _create_RV_repairs() for details.

5. Damaged component proportions that trigger a red tag

Describe the uncertainty in the amount of damaged components needed to trigger a red tag for the building. All Fragility Groups are handled in the same multivariate distribution. Consequently, correlation between various groups of component proportion limits can be specified. See _create_RV_red_tags() for details.

6. Injuries

Describe the uncertainty in the proportion of people in the affected area getting injuries exceeding a certain level of severity. FEMA P58 uses two severity levels: injury and fatality. Both levels for all Fragility Groups are handled in the same multivariate distribution. Consequently, correlation between various groups of component injury expectations can be specified. See _create_RV_injuries() for details.

define_loss_model(*self*)

Create the stochastic loss model based on the inputs provided earlier.

Following the FEMA P58 methodology, the components specified in the Damage and Loss input file are used to create Fragility Groups. Each Fragility Group corresponds to a component that might be present in the building at several locations. See _create_fragility_groups() for more details about the creation of Fragility Groups.

calculate_damage(*self*)

Characterize the damage experienced in each random event realization.

First, the time of the event (month, weekday/weekend, hour) is randomly generated for each realization. Given the event time, the number of people present at each floor of the building is calculated.

Second, the realizations that led to collapse are filtered. See _calc_collapses() for more details on collapse estimation.

Finally, the realizations that did not lead to building collapse are further investigated and the quantities of components in each damage state are estimated. See _calc_damage() for more details on damage estimation.

calculate_losses(*self*)

Characterize the consequences of damage in each random event realization.

For the sake of efficiency, only the decision variables requested in the input file are estimated. The following consequences are handled by this method:

Reconstruction time and cost Estimate the irrepairable cases based on residual drift magnitude and the provided irrepairable drift limits. Realizations that led to irrepairable damage or collapse are assigned the replacement cost and time of the building when reconstruction cost and time is estimated. Repairable cases get a cost and time estimate for each Damage State in each Performance Group. For more information about estimating irrepairability see _calc_irrepairable() and reconstruction cost and time see _calc_repair_cost_and_time() methods.

Injuries Collapse-induced injuries are based on the collapse modes and corresponding injury characterization. Injuries conditioned on no collapse are based on the affected area and the probability of injuries of various severity specified in the component data file. For more information about estimating injuries conditioned on collapse and no collapse, see _calc_collapse_injuries() and _calc_non_collapse_injuries, respectively.

Red Tag The probability of getting an unsafe placard or red tag is a function of the amount of damage experienced in various Damage States for each Performance Group. The damage limits that trigger an

unsafe placard are specified in the component data file. For more information on assigning red tags to realizations see the _calc_red_tag() method.

aggregate_results(*self*)

write_outputs(*self*)

class pelicun.control.**HAZUS_Assessment**(*hazard='EQ'*, *inj_lvls=4*)
Bases: *pelicun.control.Assessment*

An Assessment class that implements the damage and loss assessment method following the HAZUS Technical Manual and the HAZUS software.

Parameters

hazard: {'EQ', 'HU'} Identifies the type of hazard. EQ corresponds to earthquake, HU corresponds to hurricane. default: 'EQ'.

inj_lvls: int Defines the discretization used to describe the severity of injuries. The HAZUS earthquake methodology uses 4 levels. default: 4

Attributes

beta_tot Calculate the total additional uncertainty for post processing.

Methods

aggregate_results(self)	
calculate_damage(self)	Characterize the damage experienced in each random event realization.
calculate_losses(self)	Characterize the consequences of damage in each random event realization.
define_loss_model(self)	Create the stochastic loss model based on the inputs provided earlier.
define_random_variables(self)	Define the random variables used for loss assessment.
read_inputs(self, path_DL_input, path_EDP_input)	Read and process the input files to describe the loss assessment task.
write_outputs(self)	Export the results.

read_inputs(*self*, *path_DL_input*, *path_EDP_input*, *verbose=False*)
Read and process the input files to describe the loss assessment task.

Parameters

path_DL_input: string Location of the Damage and Loss input file. The file is expected to be a JSON with data stored in a standard format described in detail in the Input section of the documentation.

path_EDP_input: string Location of the EDP input file. The file is expected to follow the output formatting of Dakota. The Input section of the documentation provides more information about the expected formatting.

verbose: boolean, default: False If True, the method echoes the information read from the files. This can be useful to ensure that the information in the file is properly read by the method.

define_random_variables(*self*)
Define the random variables used for loss assessment.

Following the HAZUS methodology, only the groups of parameters below are considered random. Correlations within groups are not considered because each Fragility Group has only one Performance Group with a in this implementation.

1. Demand (EDP) distribution

Describe the uncertainty in the demands. Unlike other random variables, the EDPs are characterized by the EDP input data provided earlier. All EDPs are handled in one multivariate lognormal distribution. If more than one sample is provided, the distribution is fit to the EDP data. Otherwise, the provided data point is assumed to be the median value and the additional uncertainty prescribed describes the dispersion. See _create_RV_demands() for more details.

2. Fragility EDP limits

Describe the uncertainty in the EDP limit that corresponds to exceedance of each Damage State. EDP limits are grouped by Fragility Groups. See _create_RV_fragilities() for details.

define_loss_model (*self*)
Create the stochastic loss model based on the inputs provided earlier.

Following the HAZUS methodology, the component assemblies specified in the Damage and Loss input file are used to create Fragility Groups. Each Fragility Group corresponds to one assembly that represents every component of the given type in the structure. See _create_fragility_groups() for more details about the creation of Fragility Groups.

calculate_damage (*self*)
Characterize the damage experienced in each random event realization.

First, the time of the event (month, weekday/weekend, hour) is randomly generated for each realization. Given the event time, the number of people present at each floor of the building is calculated.

Next, the quantities of components in each damage state are estimated. See _calc_damage() for more details on damage estimation.

calculate_losses (*self*)
Characterize the consequences of damage in each random event realization.

For the sake of efficiency, only the decision variables requested in the input file are estimated. The following consequences are handled by this method for a HAZUS assessment:

Reconstruction time and cost Get a cost and time estimate for each Damage State in each Performance Group. For more information about estimating reconstruction cost and time see _calc_repair_cost_and_time() methods.

Injuries The number of injuries are based on the probability of injuries of various severity specified in the component data file. For more information about estimating injuries _calc_non_collapse_injuries.

aggregate_results (*self*)

4.1.3 pelicun.file_io module

This module has classes and methods that handle file input and output.

Contents

read_SimCenter_DL_input(input_path[, ...])	Read the damage and loss input information from a json file.

Continued on next page

Table 5 – continued from previous page

read_SimCenter_EDP_input(input_path[, ...])	Read the EDP input information from a text file with a tabular structure.
read_population_distribution(path_POP, occupancy)	Read the population distribution from an external json file.
read_component_DL_data(path_CMP, comp_info)	Read the damage and loss data for the components of the asset.
convert_P58_data_to_json(data_dir, target_dir)	Create JSON data files from publicly available P58 data.
create_HAZUS_EQ_json_files(data_dir, target_dir)	Create JSON data files from publicly available HAZUS data.
create_HAZUS_HU_json_files(data_dir, target_dir)	Create JSON data files from publicly available HAZUS data.
write_SimCenter_DL_output(output_path, output_df)	
write_SimCenter_DM_output(DM_file_path, DMG_df)	
write_SimCenter_DV_output(DV_file_path, ...)	

pelicun.file_io.**read_SimCenter_DL_input**(*input_path*, *assessment_type='P58'*, *verbose=False*)

Read the damage and loss input information from a json file.

The SimCenter in the function name refers to having specific fields available in the file. Such a file is automatically prepared by the SimCenter PBE Application, but it can also be easily manipulated or created manually. The accepted input fields are explained in detail in the Input section of the documentation.

Parameters

input_path: string Location of the DL input json file.

assessment_type: {'P58', 'HAZUS_EQ', 'HAZUS_HU'} Tailors the warnings and verifications towards the type of assessment. default: 'P58'.

verbose: boolean If True, the function echoes the information read from the file. This can be useful to ensure that the information in the file is properly read by the method.

Returns

data: dict A dictionary with all the damage and loss data.

pelicun.file_io.**read_SimCenter_EDP_input**(*input_path*, *EDP_kinds=('PID', 'PFA')*, *units={'PFA': 1.0, 'PID': 1.0}*, *verbose=False*)

Read the EDP input information from a text file with a tabular structure.

The SimCenter in the function name refers to having specific columns available in the file. Currently, the expected formatting follows the output formatting of Dakota that is applied for the dakotaTab.out. When using pelicun with the PBE Application, such a dakotaTab.out is automatically generated. The Input section of the documentation provides more information about the expected formatting of the EDP input file.

Parameters

input_path: string Location of the EDP input file.

EDP_kinds: tuple of strings, default: ('PID', 'PFA') Collection of the kinds of EDPs in the input file. The default pair of 'PID' and 'PFA' can be replaced or extended by any other EDPs.

units: dict, default: {'PID':1., 'PFA':1} Defines the unit conversion that shall be applied to the EDP values.

verbose: boolean If True, the function echoes the information read from the file. This can be useful to ensure that the information in the file is properly read by the method.

Returns

data: dict A dictionary with all the EDP data.

pelicun.file_io.**read_population_distribution**(*path_POP*, *occupancy*, *assessment_type='P58'*, *verbose=False*)

Read the population distribution from an external json file.

The population distribution is expected in a format used in FEMA P58, but the list of occupancy categories can be modified and/or extended beyond those available in that document. The population distributions for the occupancy categories from FEMA P58 and HAZUS MH are provided with pelicun in the population.json files in the corresponding folder under resources.

Note: Population distributions in HAZUS do not have a 1:1 mapping to the occupancy types provided in the Technical Manual. We expect inputs to follow the naming convention in the HAZUS Technical Manual and convert those to the broader categories here automatically. During conversion, the following assumptions are made about the occupancy classes: i) RES classes are best described as Residential; ii) COM and REL as Commercial; iii) EDU as Educational; iv) IND and AGR as Industrial; v) Hotels do not have a matching occupancy class.

Parameters

path_POP: string Location of the population distribution json file.

occupancy: string Identifies the occupancy category.

assessment_type: {'P58', 'HAZUS_EQ'} Tailors the warnings and verifications towards the type of assessment. default: 'P58'.

verbose: boolean If True, the function echoes the information read from the file. This can be useful to ensure that the information in the file is properly read by the method.

Returns

data: dict A dictionary with the population distribution data.

pelicun.file_io.**read_component_DL_data**(*path_CMP*, *comp_info*, *assessment_type='P58'*, *verbose=False*)

Read the damage and loss data for the components of the asset.

DL data for each component is assumed to be stored in a JSON file following the DL file format specified by SimCenter. The name of the file is the ID (key) of the component in the comp_info dictionary. Besides the filename, the comp_info dictionary is also used to get other pieces of data about the component that is not available in the JSON files. Therefore, the following attributes need to be provided in the comp_info: ['quantities', 'csg_weights', 'dirs', 'kind', 'distribution', 'cov', 'unit', 'locations'] Further information about these attributes is available in the Input section of the documentation.

Parameters

path_CMP: string Location of the folder that contains the component data in JSON files.

comp_info: dict Dictionary with additional information about the components.

assessment_type: {'P58', 'HAZUS_EQ', 'HAZUS_HU'} Tailors the warnings and verifications towards the type of assessment. default: 'P58'.

verbose: boolean If True, the function echoes the information read from the files. This can be useful to ensure that the information in the files is properly read by the method.

Returns

data: dict A dictionary with damage and loss data for each component.

pelicun.file_io.**convert_P58_data_to_json**(*data_dir*, *target_dir*)
Create JSON data files from publicly available P58 data.

FEMA P58 damage and loss information is publicly available in an Excel spreadsheet and also in a series of XML files as part of the PACT tool. Those files are copied to the resources folder in the pelicun repo. Here we collect the available information on Fragility Groups from those files and save the damage and loss data in the common SimCenter JSON format.

A large part of the Fragility Groups in FEMA P58 do not have complete damage and loss information available. These FGs are clearly marked with an incomplete flag in the JSON file and the 'Undefined' value highlights the missing pieces of information.

Parameters

data_dir: string Path to the folder with the FEMA P58 Excel file and a 'DL xml' subfolder in it that contains the XML files.

target_dir: string Path to the folder where the JSON files shall be saved.

pelicun.file_io.**create_HAZUS_EQ_json_files**(*data_dir*, *target_dir*)
Create JSON data files from publicly available HAZUS data.

HAZUS damage and loss information is publicly available in the technical manuals. The relevant tables have been converted into a JSON input file (hazus_data_eq.json) that is stored in the 'resources/HAZUS MH 2.1' folder in the pelicun repo. Here we read that file (or a file of similar format) and produce damage and loss data for Fragility Groups in the common SimCenter JSON format.

HAZUS handles damage and losses at the assembly level differentiating only structural and two types of non-structural component assemblies. In this implementation we consider each of those assemblies a Fragility Group and describe their damage and its consequences in a FEMA P58-like framework but using the data from the HAZUS Technical Manual.

Parameters

data_dir: string Path to the folder with the hazus_data_eq JSON file.

target_dir: string Path to the folder where the results shall be saved. The population distribution file will be saved here, the DL JSON files will be saved to a 'DL json' subfolder.

pelicun.file_io.**create_HAZUS_HU_json_files**(*data_dir*, *target_dir*)
Create JSON data files from publicly available HAZUS data.

HAZUS damage and loss information is publicly available in the technical manuals and the HAZUS software tool. The relevant data have been collected in a series of Excel files (e.g., hu_Wood.xlsx) that are stored in the 'resources/HAZUS MH 2.1 hurricane' folder in the pelicun repo. Here we read that file (or a file of similar format) and produce damage and loss data for Fragility Groups in the common SimCenter JSON format.

The HAZUS hurricane methodology handles damage and losses at the assembly level. In this implementation each building is represented by one Fragility Group that describes the damage states and their consequences in a FEMA P58-like framework but using the data from the HAZUS Technical Manual.

Note: HAZUS calculates lossess independently of damage using peak wind gust speed as a controlling variable. We fitted a model to the curves in HAZUS that assigns losses to each damage state and determines losses as a function of building damage. Results shall be in good agreement with those of HAZUS for the majority of building configurations. Exceptions and more details are provided in the ... section of the documentation.

Parameters

data_dir: string Path to the folder with the hazus_data_eq JSON file.

target_dir: string Path to the folder where the results shall be saved. The population distribution file will be saved here, the DL JSON files will be saved to a 'DL json' subfolder.

4.1.4 pelicun.model module

This module has classes and methods that define and access the model used for loss assessment.

Contents

FragilityFunction(EDP_limit)	Describes the relationship between asset response and damage.
ConsequenceFunction(DV_median, DV_distribution)	Describes the relationship between damage and a decision variable.
DamageState(ID[, weight, description, ...])	Characterizes one type of damage that corresponds to a particular DSG.
DamageStateGroup(ID, DS_set, DS_set_kind)	A set of similar component damages that are controlled by the same EDP.
PerformanceGroup(ID, location, quantity, ...)	A group of similar components that experience the same demands.
FragilityGroup(ID, demand_type, ...[, ...])	Groups a set of similar components from a loss-assessment perspective.
prep_constant_median_DV(median)	Returns a constant median Decision Variable (DV) function.
prep_bounded_linear_median_DV(median_max, ...)	Returns a bounded linear median Decision Variable (DV) function.

class pelicun.model.**FragilityFunction**(*EDP_limit*)

Bases: object

Describes the relationship between asset response and damage.

Asset response is characterized by a Demand value that represents an engineering demand parameter (EDP). Only a scalar EDP is supported currently. The damage is characterized by a set of DamageStateGroup (DSG) objects. For each DSG, the corresponding EDP limit (i.e. the EDP at which the asset is assumed to experience damage described by the DSG) is considered uncertain; hence, it is described by a random variable. The random variables that describe EDP limits for the set of DSGs are not independent.

We assume that the EDP limit will be approximated by a normal or lognormal distribution for each DSG and these variables together form a multivariate normal distribution. Following common practice, the correlation between variables is assumed perfect by default, but the framework allows the users to explore other, more realistic options.

Parameters

EDP_limit: RandomVariableSubset A multidimensional random variable that might be defined as a subset of a bigger correlated group of variables or a complete set of variables created only for this Fragility Function (FF). The number of dimensions shall be equal to the number of DSGs handled by the FF.

Methods

DSG_given_EDP(self, EDP[, force_resampling])	Given an EDP, get a damage level based on the fragility function.
P_exc(self, EDP, DSG_ID)	Return the probability of damage exceedance.

P_exc(*self*, *EDP*, *DSG_ID*)
Return the probability of damage exceedance.

Calculate the probability of exceeding the damage corresponding to the DSG identified by the DSG_ID conditioned on a particular EDP value.

Parameters

EDP: float scalar or ndarray Single EDP or numpy array of EDP values.

DSG_ID: int Identifies the conditioning DSG. The DSG numbering is 1-based, because zero typically corresponds to the undamaged state.

Returns

P_exc: float scalar or ndarray DSG exceedance probability at the given EDP point(s).

DSG_given_EDP(*self*, *EDP*, *force_resampling=False*)
Given an EDP, get a damage level based on the fragility function.

The damage is evaluated by sampling the joint distribution of fragilities corresponding to all possible damage levels and checking which damage level the given EDP falls into. This approach allows for efficient damage state evaluation for a large number of EDP realizations.

Parameters

EDP: float scalar or ndarray or Series Single EDP, or numpy array or pandas Series of EDP values.

force_resampling: bool, optional, default: False If True, the probability distribution is re-sampled before evaluating the damage for each EDP. This is not recommended if the fragility functions are correlated with other sources of uncertainty because those variables will also be resampled in this case. If False, which is the default approach, we assume that the random variable has already been sampled and the number of samples greater or equal to the number of EDP values.

Returns

DSG_ID: Series Identifies the damage that corresponds to the given EDP. A DSG_ID of 0 means no damage.

pelicun.model.**prep_constant_median_DV**(*median*)
Returns a constant median Decision Variable (DV) function.

Parameters

median: float The median DV for a consequence function with fixed median.

Returns

f: callable A function that returns the constant median DV for all component quantities.

pelicun.model.**prep_bounded_linear_median_DV**(*median_max*, *median_min*, *quantity_lower*, *quantity_upper*)
Returns a bounded linear median Decision Variable (DV) function.

The median DV equals the min and max values when the quantity is outside of the prescribed quantity bounds. When the quantity is within the bounds, the returned median is calculated by a linear function with a negative slope between max and min values.

Parameters

median_max: float, optional

median_min: float, optional Minimum and maximum limits that define the bounded_linear median DV function.

quantity_lower: float, optional

quantity_upper: float, optional Lower and upper bounds of component quantity that define the bounded_linear median DV function.

Returns

f: callable A function that returns the median DV given the quantity of damaged components.

class pelicun.model.**ConsequenceFunction**(*DV_median*, *DV_distribution*)
Bases: object

Describes the relationship between damage and a decision variable.

Indicates the distribution of a quantified Decision Variable (DV) conditioned on a component, an element, or the system reaching a given damage state (DS). DV can be reconstruction cost, repair time, casualties, injuries, etc. Its distribution might depend on the quantity of damaged components.

Parameters

DV_median: callable Describes the median DV as an f(quantity) function of the total quantity of damaged components. Use the prep_constant_median_DV, and prep_bounded_linear_median_DV helper functions to conveniently prescribe the typical FEMA P-58 functions.

DV_distribution: RandomVariableSubset A one-dimensional random variable (or a one-dimensional subset of a multi-dimensional random variable) that characterizes the uncertainty in the DV. The distribution shall be normalized by the median DV (i.e. the RVS is expected to have a unit median). Truncation can be used to prescribe lower and upper limits for the DV, such as the (0,1) domain needed for red tag evaluation.

Methods

median(self[, quantity])	Return the value of the median DV.
sample_unit_DV(self[, quantity, ...])	Sample the decision variable quantity per component unit.

median(*self*, *quantity=None*)
Return the value of the median DV.

The median DV corresponds to the component damage state (DS). If the damage consequence depends on the quantity of damaged components, the total quantity of damaged components shall be specified through the quantity parameter.

Parameters

quantity: float scalar or ndarray, optional Total quantity of damaged components that determines the magnitude of median DV. Not needed for consequence functions with a fixed median DV.

Returns

median: float scalar or ndarray A single scalar for fixed median; a scalar or an array depending on the shape of the quantity parameter for bounded_linear median.

sample_unit_DV(*self*, *quantity=None*, *sample_size=1*, *force_resampling=False*)
Sample the decision variable quantity per component unit.

The Unit Decision Variable (UDV) corresponds to the component Damage State (DS). It shall be multiplied by the quantity of damaged components to get the total DV that corresponds to the quantity of the damaged

components in the asset. If the DV depends on the total quantity of damaged components, that value shall be specified through the quantity parameter.

Parameters

quantity: float scalar, ndarray or Series, optional, default: None Total quantity of damaged components that determines the magnitude of median DV. Not needed for consequence functions with a fixed median DV.

sample_size: int, optional, default: 1 Number of samples drawn from the DV distribution. The default value yields one sample. If quantity is an array with more than one element, the sample_size parameter is ignored.

force_resampling: bool, optional, default: False If True, the DV distribution (and the corresponding RV if there are correlations) is resampled even if there are samples already available. This is not recommended if the DV distribution is correlated with other sources of uncertainty because those variables will also be resampled in this case. If False, which is the default approach, we assume that the random variable has already been sampled and the number of samples is greater or equal to the number of samples requested.

Returns

unit_DV: float scalar or ndarray Unit DV samples.

class pelicun.model.**DamageState**(*ID*, *weight=1.0*, *description=''*, *repair_cost_CF=None*, *reconstruction_time_CF=None*, *injuries_CF_set=None*, *affected_area=0.0*, *red_tag_CF=None*)

Bases: object

Characterizes one type of damage that corresponds to a particular DSG.

The occurrence of damage is evaluated at the DSG. The DS describes one of the possibly several types of damages that belong to the same DSG and the consequences of such damage.

Parameters

ID:int

weight: float, optional, default: 1.0 Describes the probability of DS occurrence, conditioned on the damage being in the DSG linked to this DS. This information is only used for DSGs with multiple DS corresponding to them. The weights of the set of DS shall sum up to 1.0 if they are mutually exclusive. When the set of DS occur simultaneously, the sum of weights typically exceeds 1.0.

description: str, optional Provides a short description of the damage state.

affected_area: float, optional, default: 0. Defines the area over which life safety hazards from this DS exist.

repair_cost_CF: ConsequenceFunction, optional A consequence function that describes the cost necessary to restore the component to its pre-disaster condition.

reconstruction_time_CF: ConsequenceFunction, optional A consequence function that describes the time, necessary to repair the damaged component to its pre-disaster condition.

injuries_CF_set: ConsequenceFunction array, optional A set of consequence functions; each describes the number of people expected to experience injury of a particular severity when the component is in this DS. Any number of injury-levels can be considered.

red_tag_CF: ConsequenceFunction, optional A consequence function that describes the proportion of components (within a Performance Group) that needs to be damaged to trigger an unsafe placard (i.e. red tag) for the building during post-disaster inspection.

Attributes

description Return the damage description.

weight Return the weight of DS among the set of damage states in the DSG.

Methods

red_tag_dmg_limit(self[, sample_size])	Sample the red tag consequence function and return the proportion of components that needs to be damaged to trigger a red tag.
unit_injuries(self[, severity_level, ...])	Sample the injury consequence function that corresponds to the specified level of severity and return the injuries per component unit.
unit_reconstruction_time(self[, quantity, ...])	Sample the reconstruction time distribution and return the unit reconstruction times.
unit_repair_cost(self[, quantity, sample_size])	Sample the repair cost distribution and return the unit repair costs.

property description
Return the damage description.

property weight
Return the weight of DS among the set of damage states in the DSG.

unit_repair_cost (*self*, *quantity=None*, *sample_size=1*, ***kwargs*)
Sample the repair cost distribution and return the unit repair costs.

The unit repair costs shall be multiplied by the quantity of damaged components to get the total repair costs for the components in this DS.

Parameters

quantity: float scalar, ndarray or Series, optional, default: None Total quantity of damaged components that determines the median repair cost. Not used for repair cost models with fixed median.

sample_size: int, optional, default: 1 Number of samples drawn from the repair cost distribution. The default value yields one sample.

Returns

unit_repair_cost: float scalar or ndarray Unit repair cost samples.

unit_reconstruction_time (*self*, *quantity=None*, *sample_size=1*, ***kwargs*)
Sample the reconstruction time distribution and return the unit reconstruction times.

The unit reconstruction times shall be multiplied by the quantity of damaged components to get the total reconstruction time for the components in this DS.

Parameters

quantity: float scalar, ndarray or Series, optional, default: None Total quantity of damaged components that determines the magnitude of median reconstruction time. Not used for reconstruction time models with fixed median.

sample_size: int, optional, default: 1 Number of samples drawn from the reconstruction time distribution. The default value yields one sample.

Returns

unit_reconstruction_time: float scalar or ndarray Unit reconstruction time samples.

red_tag_dmg_limit(*self*, *sample_size=1*, ***kwargs*)
Sample the red tag consequence function and return the proportion of components that needs to be damaged to trigger a red tag.

The red tag consequence function is assumed to have a fixed median value that does not depend on the quantity of damaged components.

Parameters

sample_size: int, optional, default: 1 Number of samples drawn from the red tag consequence distribution. The default value yields one sample.

Returns

red_tag_trigger: float scalar or ndarray Samples of damaged component proportions that trigger a red tag.

unit_injuries(*self*, *severity_level=0*, *sample_size=1*, ***kwargs*)
Sample the injury consequence function that corresponds to the specified level of severity and return the injuries per component unit.

The injury consequence function is assumed to have a fixed median value that does not depend on the quantity of damaged components (i.e. the number of injuries per component unit does not change with the quantity of components.)

Parameters

severity_level: int, optional, default: 1 Identifies which injury consequence to sample. The indexing of severity levels is zero-based.

sample_size: int, optional, default: 1 Number of samples drawn from the injury consequence distribution. The default value yields one sample.

Returns

unit_injuries: float scalar or ndarray Unit injury samples.

class pelicun.model.**DamageStateGroup**(*ID*, *DS_set*, *DS_set_kind*)
Bases: object

A set of similar component damages that are controlled by the same EDP.

Damages are described in detail by the set of Damage State objects. Damages in a DSG are assumed to occur at the same EDP magnitude. A Damage State Group (DSG) might have only a single DS in the simplest case.

Parameters

ID: int

DS_set: DamageState array

DS_set_kind: {'single', 'mutually_exclusive', 'simultaneous'} Specifies the relationship among the DS in the set. When only one DS is defined, use the 'single' option to improve calculation efficiency. When multiple DS are present, the 'mutually_exclusive' option assumes that the occurrence of one DS precludes the occurrence of another DS. In such a case, the weights of the DS in the set shall sum up to 1.0. In a 'simultaneous' case the DS are independent and unrelated. Hence, they can occur at the same time and at least one of them has to occur.

class pelicun.model.**PerformanceGroup**(*ID, location, quantity, fragility_functions, DSG_set, csg_weights=[1.0], direction=0*)
Bases: object

A group of similar components that experience the same demands.

FEMA P-58: Performance Groups (PGs) are a sub-categorization of fragility groups. A performance group is a subset of fragility group components that are subjected to the same demands (e.g. story drift, floor acceleration, etc.).

In buildings, most performance groups shall be organized by story level. There is no need to separate performance groups by direction, because the direction of components within a group can be specified during definition, and it will be taken into consideration in the analysis.

Parameters

ID: int

location: int Identifies the location of the components that belong to the PG. In a building, location shall typically refer to the story of the building. The location assigned to each PG shall be in agreement with the locations assigned to the Demand objects.

quantity: RandomVariableSubset Specifies the quantity of components that belong to this PG. Uncertainty in component quantities is considered by assigning a random variable to this property.

fragility_functions: FragilityFunction list Each fragility function describes the probability that the damage in a subset of components will meet or exceed the damages described by each damage state group in the DSG_set. Each is a multi-dimensional function if there is more than one DSG. The number of functions shall match the number of subsets defined by the *csg_weights* parameter.

DSG_set: DamageStateGroup array A set of sequential Damage State Groups that describe the plausible set of damage states of the components in the FG.

csg_weights: float ndarray, optional, default: [1.0] Identifies subgroups of components within a PG, each of which have perfectly correlated behavior. Correlation between the damage and consequences among subgroups is controlled by the *correlation* parameter of the FragilityGroup that the PG belongs to. Note that if the components are assumed to have perfectly correlated behavior at the PG level, assigning several subgroups to the PG is unnecessary. This input shall be a list of weights that are applied to the quantity of components to define the amount of components in each subgroup. The sum of assigned weights shall be 1.0.

directions: int ndarray, optional, default: [0] Identifies the direction of each subgroup of components within the PG. The number of directions shall be identical to the number of csg_weights assigned. In buildings, directions typically correspond to the orientation of components in plane. Hence, using 0 or 1 to identify 'X' or 'Y' is recommended. These directions shall be in agreement with the directions assigned to Demand objects.

Methods

P_exc(self, EDP, DSG_ID)	This is a convenience function that provides a shortcut to fragility_function.P_exc().

P_exc (*self*, *EDP*, *DSG_ID*)

This is a convenience function that provides a shortcut to fragility_function.P_exc(). It calculates the exceedance probability of a given DSG conditioned on the provided EDP value(s). The fragility functions assigned to the first subset are used for this calculation because P_exc shall be identical among subsets.

Parameters

EDP: float scalar or ndarray Single EDP or numpy array of EDP values.

DSG_ID: int Identifies the DSG of interest.

Returns

P_exc: float scalar or ndarray Exceedance probability of the given DSG at the EDP point(s).

class pelicun.model.**FragilityGroup**(*ID*, *demand_type*, *performance_groups*, *directional=True*, *correlation=True*, *demand_location_offset=0*, *incomplete=False*, *name=''*, *description=''*)

Bases: object

Groups a set of similar components from a loss-assessment perspective.

Characterizes a set of structural or non-structural components that have similar construction characteristics, similar potential modes of damage, similar probability of incurring those modes of damage, and similar potential consequences resulting from their damage.

Parameters

ID: int

demand_type: {'PID', 'PFA', 'PSD', 'PSA', 'ePGA', 'PGD'} The type of Engineering Demand Parameter (EDP) that controls the damage of the components in the FG. See Demand for acronym descriptions.

performance_groups: PerformanceGroup array A list of performance groups that contain the components characterized by the FG.

directional: bool, optional, default: True Determines whether the components in the FG are sensitive to the directionality of the EDP.

correlation: bool, optional, default: True Determines whether the components within a Performance Group (PG) will have correlated or uncorrelated damage. Correlated damage means that all components will have the same damage state. In the uncorrelated case, each component in the performance group will have its damage state evaluated independently. Correlated damage reduces the required computational effort for the calculation. Incorrect correlation modeling will only slightly affect the mean estimates, but might significantly change the dispersion of results.

demand_location_offset: int, optional, default: 0 Indicates if the location for the demand shall be different from the location of the components. Damage to components of the ceiling, for example, is controlled by demands on the floor above the one that the components belong to. This can be indicated by setting the demand_location_offset to 1 for such an FG.

incomplete: bool, optional, default: False Indicates that the FG information is not complete and corresponding results shall be treated with caution.

name: str, optional, default: '' Provides a short description of the fragility group.

description: str, optional, default: '' Provides a detailed description of the fragility group.

Attributes

description Return the fragility group description.

name Return the name of the fragility group.

property description
Return the fragility group description.

property name
Return the name of the fragility group.

4.1.5 pelicun.uq module

This module defines constants, classes and methods for uncertainty quantification in pelicun.

Contents

RandomVariable(ID, dimension_tags[, ...])	Characterizes a Random Variable (RV) that represents a source of uncertainty in the calculation.
RandomVariableSubset(RV, tags)	Provides convenient access to a subset of components of a RandomVariable.
tmvn_rvs(mu, COV[, lower, upper, size])	Sample a truncated MVN distribution.
mvn_orthotope_density(mu, COV[, lower, upper])	Estimate the probability density within a hyperrectangle for an MVN distr.
tmvn_MLE(samples[, tr_lower, tr_upper, ...])	Fit a truncated multivariate normal distribution to samples using MLE.

pelicun.uq.**tmvn_rvs**(*mu*, *COV*, *lower=None*, *upper=None*, *size=1*)

Sample a truncated MVN distribution.

Truncation of the multivariate normal distribution is currently considered through rejection sampling. The applicability of this method is limited by the amount of probability density enclosed by the hyperrectangle defined by the truncation limits. The lower that density is, the more samples will need to be rejected which makes the method inefficient when the tails of the MVN shall be sampled in high-dimensional space. Such cases can be handled by a Gibbs sampler, which is a planned future feature of this function.

Parameters

mu: float scalar or ndarray Mean(s) of the non-truncated distribution.

COV: float ndarray Covariance matrix of the non-truncated distribution.

lower: float vector, optional, default: None Lower bound(s) for the truncated distributions. A scalar value can be used for a univariate case, while a list of bounds is expected in multivariate cases. If the distribution is non-truncated from below in a subset of the dimensions, assign an infinite value (i.e. -numpy.inf) to those dimensions.

upper: float vector, optional, default: None Upper bound(s) for the truncated distributions. A scalar value can be used for a univariate case, while a list of bounds is expected in multivariate cases. If the distribution is non-truncated from above in a subset of the dimensions, assign an infinite value (i.e. numpy.inf) to those dimensions.

size: int Number of samples requested.

Returns

samples: float ndarray Samples generated from the truncated distribution.

pelicun.uq.**mvn_orthotope_density**(*mu*, *COV*, *lower=None*, *upper=None*)

Estimate the probability density within a hyperrectangle for an MVN distr.

Use the method of Alan Genz (1992) to estimate the probability density of a multivariate normal distribution within an n-orthotope (i.e., hyperrectangle) defined by its lower and upper bounds. Limits can be relaxed in any direction by assigning infinite bounds (i.e. numpy.inf).

Parameters

mu: float scalar or ndarray Mean(s) of the non-truncated distribution.

COV: float ndarray Covariance matrix of the non-truncated distribution

lower: float vector, optional, default: None Lower bound(s) for the truncated distributions. A scalar value can be used for a univariate case, while a list of bounds is expected in multivariate cases. If the distribution is non-truncated from below in a subset of the dimensions, use either *None* or assign an infinite value (i.e. -numpy.inf) to those dimensions.

upper: float vector, optional, default: None Upper bound(s) for the truncated distributions. A scalar value can be used for a univariate case, while a list of bounds is expected in multivariate cases. If the distribution is non-truncated from above in a subset of the dimensions, use either *None* or assign an infinite value (i.e. numpy.inf) to those dimensions.

Returns

———-

alpha: float Estimate of the probability density within the hyperrectangle

eps_alpha: float Estimate of the error in alpha.

pelicun.uq.**tmvn_MLE**(*samples*, *tr_lower=None*, *tr_upper=None*, *censored_count=0*, *det_lower=None*, *det_upper=None*, *alpha_lim=None*)

Fit a truncated multivariate normal distribution to samples using MLE.

The number of dimensions of the distribution function are inferred from the shape of the sample data. Censoring is automatically considered if the number of censored samples and the corresponding detection limits are provided. Infinite or unspecified truncation limits lead to fitting a non-truncated normal distribution in that dimension.

Parameters

samples: ndarray Raw data that serves as the basis of estimation. The number of samples equals the number of columns and each row introduces a new feature. In other words: a list of sample lists is expected where each sample list is a collection of samples of one variable.

tr_lower: float vector, optional, default: None Lower bound(s) for the truncated distributions. A scalar value can be used for a univariate case, while a list of bounds is expected in multivariate cases. If the distribution is non-truncated from below in a subset of the dimensions, use either *None* or assign an infinite value (i.e. -numpy.inf) to those dimensions.

tr_upper: float vector, optional, default: None Upper bound(s) for the truncated distributions. A scalar value can be used for a univariate case, while a list of bounds is expected in multivariate cases. If the distribution is non-truncated from above in a subset of the dimensions, use either *None* or assign an infinite value (i.e. numpy.inf) to those dimensions.

censored_count: int, optional, default: None The number of censored samples that are beyond the detection limits. All samples outside the detection limits are aggregated into one set. This works the same way in one and in multiple dimensions. Prescription of specific censored sample counts for sub-regions of the input space outside the detection limits is not supported.

det_lower: float ndarray, optional, default: None Lower detection limit(s) for censored data. In multivariate cases the limits need to be defined as a vector; a scalar value is sufficient in a univariate case. If the data is not censored from below in a particular dimension, assign None to that position of the ndarray.

det_upper: float ndarray, optional, default: None Upper detection limit(s) for censored data. In multivariate cases the limits need to be defined as a vector; a scalar value is sufficient in a univariate case. If the data is not censored from above in a particular dimension, assign None to that position of the ndarray.

alpha_lim: float, optional, default:None Introduces a lower limit to the probability density within the n-orthotope defined by the truncation limits. Assigning a reasonable minimum

(such as 1e-4) can be useful when the mean of the distribution is several standard deviations from the truncation limits and the sample size is small. Such cases without a limit often converge to distant means with inflated variances. Besides being incorrect estimates, those solutions only offer negligible reduction in the negative log likelihood, while making subsequent sampling of the truncated normal distribution very challenging.

Returns

mu: float scalar or ndarray Mean of the fitted probability distribution. A vector of means is returned in a multivariate case.

COV: float scalar or 2D ndarray Covariance matrix of the fitted probability distribution. A 2D square ndarray is returned in a multi-dimensional case, while a single variance (not standard deviation!) value is returned in a univariate case.

class pelicun.uq.**RandomVariable**(*ID*, *dimension_tags*, *raw_data=None*, *detection_limits=None*, *censored_count=None*, *distribution_kind=None*, *theta=None*, *COV=None*, *corr_ref='pre'*, *p_set=None*, *truncation_limits=None*)

Bases: object

Characterizes a Random Variable (RV) that represents a source of uncertainty in the calculation.

The uncertainty can be described either through raw data or through a pre-defined distribution function. When using raw data, provide potentially correlated raw samples in a 2 dimensional array. If the data is left or right censored in any number of its dimensions, provide the list of detection limits and the number of censored samples. No other information is needed to define the object from raw data. Then, either resample the raw data, or fit a prescribed distribution to the samples and sample from that distribution later.

Alternatively, one can choose to prescribe a distribution type and its parameters and sample from that distribution later.

Parameters

ID: int

dimension_tags: str array A series of strings that identify the stochastic model parameters that correspond to each dimension of the random variable. When the RV is one dimensional, the dim_tag is a single string. In multi-dimensional cases, the order of strings shall match the order of elements provided as other inputs.

raw_data: float scalar or ndarray, optional, default: None Samples of an uncertain variable. The samples can describe a multi-dimensional random variable if they are arranged in a 2D ndarray.

detection_limits: float ndarray, optional, default: None Defines the limits for censored data. The limits need to be defined in a 2D ndarray that is structured as two vectors with N elements. The vectors collect left and right limits for the N dimensions. If the data is not censored in a particular direction, assign None to that position of the ndarray. Replacing one of the vectors with None will assign no censoring to all dimensions in that direction. The default value corresponds to no censoring in either dimension.

censored_count: int, optional, default: None The number of censored samples that are beyond the detection limits. All samples outside the detection limits are aggregated into one set. This works the same way in one and in multiple dimensions. Prescription of censored sample counts for sub-regions of the input space outside the detection limits is not yet supported. If such an approach is desired, the censored raw data shall be used to fit a distribution in a pre-processing step and the fitted distribution can be specified for this random variable.

distribution_kind: {'normal', 'lognormal', 'multinomial'}, optional, default: None Defines the type of probability distribution when raw data is not provided, but the distri-

bution is directly specified. When part of the data is normal in log space, while the other part is normal in linear space, define a list of distribution tags such as ['normal', 'normal', 'lognormal']. Make sure that the covariance matrix is based on log transformed data for the lognormally distributed variables! Mixing normal distributions with multinomials is not supported.

theta: float scalar or ndarray, optional, default: None Median of the probability distribution. A vector of medians is expected in a multi-dimensional case.

COV: float scalar or 2D ndarray, optional, default: None Covariance matrix of the random variable. In a multi-dimensional case this parameter has to be a 2D square ndarray, and the number of its rows has to be equal to the number of elements in the supplied theta vector. In a one-dimensional case, a single value is expected that equals the variance (not the standard deviation!) of the distribution. The COV for lognormal variables is assumed to be specified in logarithmic space.

corr_ref: {'pre', 'post'}, optional, default: 'pre' Determines whether the correlations prescribed by the covariance matrix refer to the distribution functions before or after truncation. The default 'pre' setting assumes that pre-truncation correlations are prescribed and creates a multivariate normal distribution using the COV matrix. That distribution is truncated according to the prescribed truncation limits. The other option assumes that post-truncation correlations are prescribed. The post-truncation distribution is not multivariate normal in general. Currently we use a Gaussian copula to describe the dependence between the truncated variables. Similarly to other characteristics, the *corr_ref* can be defined as a single string, or a vector of strings. The former assigns the same option to all dimensions, while the latter allows for more flexible assignment by setting the corr_ref for each dimension individually.

p_set: float vector, optional, default: None Probabilities of a finite set of events described by a multinomial distribution. The RV will have binomial distribution if only one element is provided in this vector. The number of events equals the number of vector elements if their probabilities sum up to 1.0. If the sum is less than 1.0, then an additional event is assumed with the remaining probability of occurrence assigned to it. The sum of event probabilities shall never be more than 1.0.

truncation_limits: float ndarray, optional, default: None Defines the limits for truncated distributions. The limits need to be defined in a 2D ndarray that is structured as two vectors with N elements. The vectors collect left and right limits for the N dimensions. If the distribution is not truncated in a particular direction, assign None to that position of the ndarray. Replacing one of the vectors with None will assign no truncation to all dimensions in that direction. The default value corresponds to no truncation in either dimension.

Attributes

COV Return the covariance matrix of the probability distribution.

censored_count Return the number of samples beyond the detection limits.

corr Return the correlation matrix of the probability distribution.

det_lower Return the lower detection limit(s) corresponding to the raw data in either linear or log space according to the distribution.

det_upper Return the upper detection limit(s) corresponding to the raw data in either linear or log space according to the distribution.

detection_limits Return the detection limits corresponding to the raw data in linear space.

dimension_tags Return the tags corresponding to the dimensions of the variable.

distribution_kind Return the assigned probability distribution family.

mu Return the mean value(s) of the probability distribution.

raw Return the pre-assigned raw data.

samples Return the pre-generated samples from the distribution.

sig Return the standard deviation vector of the probability distribution.

theta Return the median value(s) of the probability distribution.

tr_limits_post Return the *post* truncation limits of the probability distribution in linear space.

tr_limits_pre Return the *pre* truncation limits of the probability distribution in linear space.

tr_lower_post Return the lower *post* truncation limit(s) corresponding to the distribution in either linear or log space according to the distribution.

tr_lower_pre Return the lower *pre* truncation limit(s) corresponding to the distribution in either linear or log space according to the distribution.

tr_upper_post Return the upper *post* truncation limit(s) corresponding to the distribution in either linear or log space according to the distribution.

tr_upper_pre Return the upper *pre* truncation limit(s) corresponding to the distribution in either linear or log space according to the distribution.

var Return the variance vector of the probability distribution.

Methods

fit_distribution(self, distribution_kind[, ...])	Estimate the parameters of a probability distribution from raw data.
orthotope_density(self[, lower, upper])	Estimate the probability density within an orthotope for a TMVN distr.
sample_distribution(self, sample_size[, ...])	Sample the probability distribution assigned to the random variable.

property distribution_kind
Return the assigned probability distribution family.

property theta
Return the median value(s) of the probability distribution.

property mu
Return the mean value(s) of the probability distribution. Note that the mean value is in log space for lognormal distributions.

property COV
Return the covariance matrix of the probability distribution. Note that the covariances are in log space for lognormal distributions.

property corr
Return the correlation matrix of the probability distribution. Note that correlation coefficient correspond to the joint distribution in log space for lognormal distributions.

property var

Return the variance vector of the probability distribution. Note that the variances are in log space for lognormal distributions.

property sig
Return the standard deviation vector of the probability distribution. Note that the standard deviations are in log space for lognormal distributions.

property dimension_tags
Return the tags corresponding to the dimensions of the variable.

property detection_limits
Return the detection limits corresponding to the raw data in linear space.

property det_lower
Return the lower detection limit(s) corresponding to the raw data in either linear or log space according to the distribution.

property det_upper
Return the upper detection limit(s) corresponding to the raw data in either linear or log space according to the distribution.

property tr_limits_pre
Return the *pre* truncation limits of the probability distribution in linear space.

property tr_limits_post
Return the *post* truncation limits of the probability distribution in linear space.

property tr_lower_pre
Return the lower *pre* truncation limit(s) corresponding to the distribution in either linear or log space according to the distribution.

property tr_upper_pre
Return the upper *pre* truncation limit(s) corresponding to the distribution in either linear or log space according to the distribution.

property tr_lower_post
Return the lower *post* truncation limit(s) corresponding to the distribution in either linear or log space according to the distribution.

property tr_upper_post
Return the upper *post* truncation limit(s) corresponding to the distribution in either linear or log space according to the distribution.

property censored_count
Return the number of samples beyond the detection limits.

property samples
Return the pre-generated samples from the distribution.

property raw
Return the pre-assigned raw data.

fit_distribution(*self*, *distribution_kind*, *truncation_limits=None*)
Estimate the parameters of a probability distribution from raw data.

Parameter estimates are calculated using maximum likelihood estimation. If the data spans multiple dimensions, the estimates will also describe a multi-dimensional distribution automatically. Data censoring is also automatically taken into consideration following the detection limits specified previously for the random variable. Truncated target distributions can be specified through the truncation limits. The specified truncation limits are applied after the correlations are set. In other words, the corr_ref proprety of the RV is set to 'pre' when fitting a distribution.

Besides returning the parameter estimates, their values are also stored as theta and COV attributes of the RandomVariable object for future use.

Parameters

distribution_kind: {'normal', 'lognormal'} or a list of those Specifies the type of the probability distribution that is fit to the raw data. When part of the data is normal in log space, while the other part is normal in linear space, define a list of distribution tags such as ['normal', 'normal', 'lognormal'].

truncation_limits: float ndarray, optional, default: None Defines the limits for truncated distributions. The limits need to be defined in a 2D ndarray that is structured as two vectors with N elements. The vectors collect left and right limits for the N dimensions. If the distribution is not truncated in a particular direction, assign None to that position of the ndarray. Replacing one of the vectors with None will assign no truncation to all dimensions in that direction. The default value corresponds to no truncation in either dimension.

Returns

theta: float scalar or ndarray Median of the probability distribution. A vector of medians is returned in a multi-dimensional case.

COV: float scalar or 2D ndarray Covariance matrix of the probability distribution. A 2D square ndarray is returned in a multi-dimensional case.

sample_distribution(*self*, *sample_size*, *preserve_order=False*)

Sample the probability distribution assigned to the random variable.

Normal distributions (including truncated and/or multivariate normal and lognormal) are sampled using the tmvn_rvs() method in this module. If post-truncation correlations are set for a dimension, the corresponding truncations are enforced after sampling by first applying probability integral transformation to transform samples from the non-truncated normal to standard uniform distribution, and then applying inverse probability integral transformation to transform the samples from standard uniform to the desired truncated normal distribution. Multinomial distributions are sampled using the multinomial method in scipy. The samples are returned and also stored in the *sample* attribute of the RV.

If the random variable is defined by raw data only, we sample from the raw data.

Parameters

sample_size: int Number of samples requested.

preserve_order: bool, default: False Influences sampling from raw data. If True, the samples are copies of the first n rows of the raw data where n is the sample_size. This only works for sample_size <= raw data size. If False, the samples are drawn from the raw data pool with replacement.

Returns

samples: DataFrame Samples generated from the distribution. Columns correspond to the dimension tags that identify the variables.

orthotope_density(*self*, *lower=None*, *upper=None*)

Estimate the probability density within an orthotope for a TMVN distr.

Use the mvn_orthotope_density function in this module for the calculation. Pre-defined truncation limits for the RV are automatically taken into consideration. Limits for lognormal distributions shall be provided in linear space - the conversion is performed by the algorithm automatically. Pre- and post-truncation correlation is also considered automatically.

Parameters

lower: float vector, optional, default: None Lower bound(s) of the orthotope. A scalar value can be used for a univariate RV; a list of bounds is expected in multivariate cases. If the orthotope is not bounded from below in any dimension, use either 'None' or assign an infinite value (i.e. -numpy.inf) to that dimension.

upper: float vector, optional, default: None Upper bound(s) of the orthotope. A scalar value can be used for a univariate RV; a list of bounds is expected in multivariate cases. If the orthotope is not bounded from above in any dimension, use either 'None' or assign an infinite value (i.e. numpy.inf) to that dimension.

Returns

alpha: float Estimate of the probability density within the orthotope.

eps_alpha: float Estimate of the error in alpha.

class pelicun.uq.**RandomVariableSubset**(*RV*, *tags*)

Bases: object

Provides convenient access to a subset of components of a RandomVariable.

This object is useful when working with multivariate RVs, but it is used in all cases to provide a general approach.

Parameters

RV: RandomVariable The potentially multivariate random variable that is accessed through this object.

tags: str or list of str A string or list of strings that identify the subset of component we are interested in. These strings shall be among the *dimension_tags* of the RV.

Attributes

samples Return the pre-generated samples of the selected component from the RV distribution.

tags Return the tags corresponding to the components in the RV subset.

Methods

orthotope_density(self[, lower, upper])	Return the density within the orthotope in the marginal pdf of the RVS.
sample_distribution(self, sample_size[, ...])	Sample the probability distribution assigned to the connected RV.

property tags

Return the tags corresponding to the components in the RV subset.

property samples

Return the pre-generated samples of the selected component from the RV distribution.

sample_distribution(*self*, *sample_size*, *preserve_order=False*)

Sample the probability distribution assigned to the connected RV.

Note that this function will sample the full, potentially multivariate, distribution.

Parameters

sample_size: int Number of samples requested.

preserve_order: bool, default: False Influences sampling from raw data. If True, the samples are copies of the first n rows of the raw data where n is the sample_size. This only

works for sample_size <= raw data size. If False, the samples are drawn from the raw data pool with replacement.

Returns

samples: DataFrame Samples of the selected component generated from the distribution.

orthotope_density(*self*, *lower=None*, *upper=None*)

Return the density within the orthotope in the marginal pdf of the RVS.

The function considers the influence of every dependent variable in the RV on the marginal pdf of the RVS. Note that such influence only occurs when the RV is a truncated distribution and at least two variables are dependent. Pre- and post-truncation correlation is considered automatically.

Parameters

lower: float vector, optional, default: None Lower bound(s) of the orthotope. A scalar value can be used for a univariate RVS; a list of bounds is expected in multivariate cases. If the orthotope is not bounded from below in any dimension, use either 'None' or assign an infinite value (i.e. -numpy.inf) to that dimension.

upper: float vector, optional, default: None Upper bound(s) of the orthotope. A scalar value can be used for a univariate RVS; a list of bounds is expected in multivariate cases. If the orthotope is not bounded from above in any dimension, use either 'None' or assign an infinite value (i.e. numpy.inf) to that dimension.

Returns

alpha: float Estimate of the probability density within the orthotope.

eps_alpha: float Estimate of the error in alpha.

CHAPTER

FIVE

LICENSE

pelicun is distributed under the BSD 3-Clause license, see LICENSE.

CHAPTER

SIX

ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grant No. 1612843. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

CHAPTER

SEVEN

CONTACT

Adam Zsarnóczay, NHERI SimCenter, Stanford University, adamzs@stanford.edu

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

H

M

N

O

P

R

S

T

U

V

W