Monte-Carlo simulations with DELPHIN 5.8 using Python package SHARK

A Python tutorial

Astrid Tijskens

astrid.tijskens@kuleuven.be

05-12-2017

Contents

1 Introduction

To facilitate probabilistic Monte-Carlo simulations with DELPHIN 5.8, a Python package called 'SHARK' was developed. SHARK allows the user to:

- Create a sampling scheme from given parameter distributions
- Automatically change parameter values in existing DELPHIN project files and save these as new DELPHIN project files
- Automatically run the newly created DELHIN project files
- Post-process the simulation output using different damage prediction models

SHARK is able to automatically recognise which parameters need to be edited by which values. For this to work, however, some formatting rules need to be followed. This tutorial describes step-by-step how to set up the project correctly and how to use SHARK. The general work flow, based on the probabilistic methodology developed by Van Gelder et al. [2014], is as follows:

1. Create one DELPHIN project file for each design options
2. Generate sampling scheme based on the given parameters and parameter distributions
3. Generate DELPHIN project files based on the design options and sampling scheme
4. Run the simulations
5. Post-process the simulation output

These different steps will be explained in the following sections.

1.1 Installing Python and SHARK

The best way to install Python is via Anaconda. Go to `https://www.anaconda.com/download`, download the Python 3.6 version for your machine and install. After installing, open an Anaconda Command Prompt and type:

```
python -V
```

This should return the following (or something similar):

```
Python 3.6.2 :: Anaconda custom (64-bit)
```

Once Python is installed, you can install SHARK to following way. Go to `https://github.com/astridtijskens/Shark`, open the folder `~\dist\` and download the zip-file of the latest Shark version to your machine. Unzip and remove the version from the folder name, go to `~\SHARK\shark\data\` and open `dir_ext_clim.txt`. Change the path you see here to the directory where you store your climate data and save the file.
Open an Anaconda Command Prompt and change the working directory to the path of the folder `SHARK` by typing the following (replace `~\path` by the path where you saved `SHARK`):

```
cd ~\path\SHARK
```

This folder should contain a file `setup.py`. To install SHARK, type the following:

```
python setup.py install
```

If you are asked to install dependent packages, do so. If no errors come up, you should see the following line:

```
Successfully built SHARK
```

SHARK should be installed correctly now. You can check this by going to `~\Anaconda3\Lib\site-packages` or by typing the following in the Anaconda Command Prompt:

```
conda list
```

If SHARK is listed, it was installed correctly and you can proceed using SHARK.

1.2 SHARK updates and reporting bugs

New versions or updates of SHARK will become available on https://github.com/astridtijskens/Shark. If you encounter a bug or an issue, this can be reported on https://github.com/astridtijskens/Shark/issues or by sending an email to astrid.tijskens@kuleuven.be.

1.3 Examples

Enclosed in the package distribution are two examples. Example 1 will be used to explain the structure and methodology of SHARK. Example 2 starts from the same basis but includes convergence monitoring. This will be discussed in paragraph 6.

2 Creating DELPHIN project files

A probabilistic DELPHIN project has a specific folder structure. The example accompanied by this tutorial has the same folder structure. This structure should be kept as is and not changed.

```
'Project name' (e.g. Example)
├─Delphin files
│  ├─Materials
│  │  ├─'material_file.m6' (e.g. brick_001.m6)
│  │  └─...
│  └─Originals
│     ├─'design_option.dpj' (e.g. Option_000.dpj)
│     └─...
├─Sampling scheme
├─config.py
├─run.py
└─postprocess.py
```

Once the folder structure is created, the first step is to create one DELPHIN project file for each design option included in the probabilistic analysis. As this is very project specific, this step is not automated but needs to be done manually. In this tutorial, an external wall with internal insulation is studied and three design options are included: a reference wall without insulation (`Option_000.dpj`), a wall with 6 cm XPS insulation (`Option_001.dpj` and a wall with 6 cm CaSi insulation (`Option_002.dpj`. These DELPHIN project files are created as usual. For SHARK to work however, some formatting rules need to be followed. These rules are described in the remainder of this manual.

2.1 Automatically changing materials

If you want to include different materials in the probabilistic assessment, e.g. different brick types for the masonry wall, you need to load all the different materials into the DELPHIN project file, but assign only one of them to the construction, as shown in figure 1. Which one you assign does not matter. All materials need to have the same name, combined with a number or letter, e.g. 'Brick1', 'Brick2', 'Brick3' or 'insulation_A', 'insulation_B', 'insulation_C'.
Custom materials, which are not included in the DELPHIN material library, can be saved in the 'Materials' folder. To load these materials, right click in the Materials window and click on 'insert from external file'.

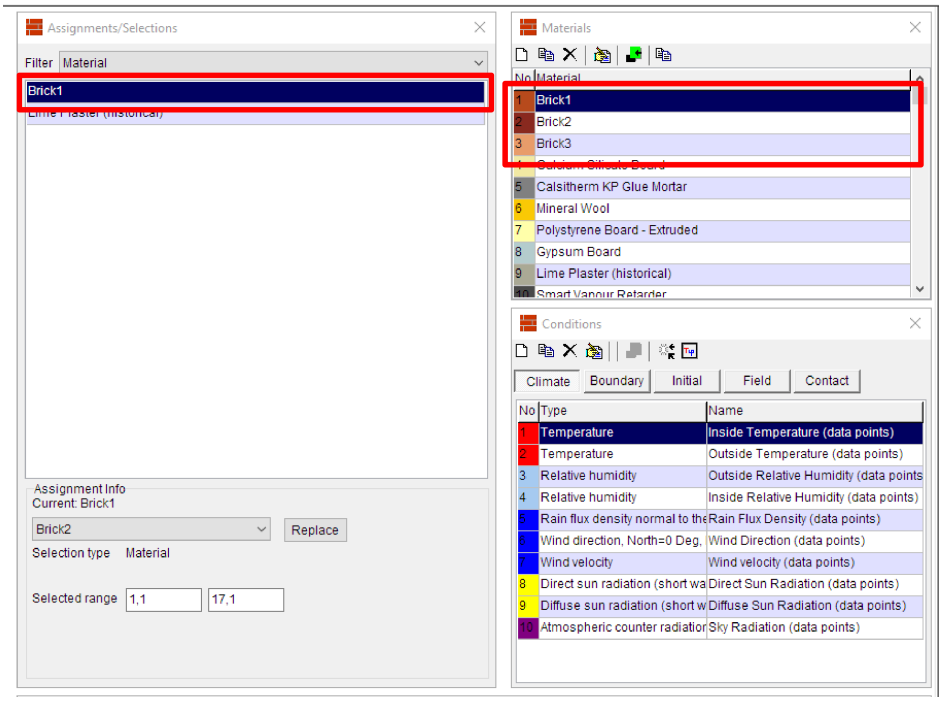

Figure 1: Three brick materials are loaded into the project file, but only one is assigned to the construction.

Note: Although it would technically be possible to change the material parameters of a specific material file, instead of using different material files, this is not supported by SHARK. This is because the parameters of a material are often not independent of each other, thus changing only one parameter might result in an unrealistic material.

2.2 Automatically changing dimensions of a building component

It is possible to automatically change the dimensions of a building component, e.g. the brick wall thickness. For this to work, the DELPHIN geometry should be discretised (as usual), except for the building component of which you want to change the dimensions. This is shown in figure 2. If your project requires outputs within this building component, this is possible by only discretising the part where you want to have an output. In the example, we want to have an output at 0.5 cm from the exterior surface (i.e. to evaluate the frost damage risk) and at 5 cm from the interior brick surface (i.e. to evaluate the wooden beams ends decay risk). Hence, the first 0.5 cm and last 5 cm of the masonry wall component are discretised (and the outputs are assigned), the section in between is not. SHARK will automatically change the dimensions of this column according to the sampled dimension values (see section 3) and subsequently discretise this column.

2.3 Automatically changing interior and exterior climate

The interior and exterior climate can automatically be changed if the climate conditions and boundary conditions are named correctly. The name of the climate conditions of the interior temperature and relative humidity or vapour pressure should contain the word 'inside'; the name of the climate conditions for the exterior temperature and relative humidity should contain the word 'outside'. Both upper and lower case (or a combination) are allowed. All other exterior climate conditions (rain, wind velocity, wind direction, direct radiation, diffuse radiation, atmospheric radiation) can be named arbitrarily. The same rules apply for the boundary conditions, as shown in figure 3.

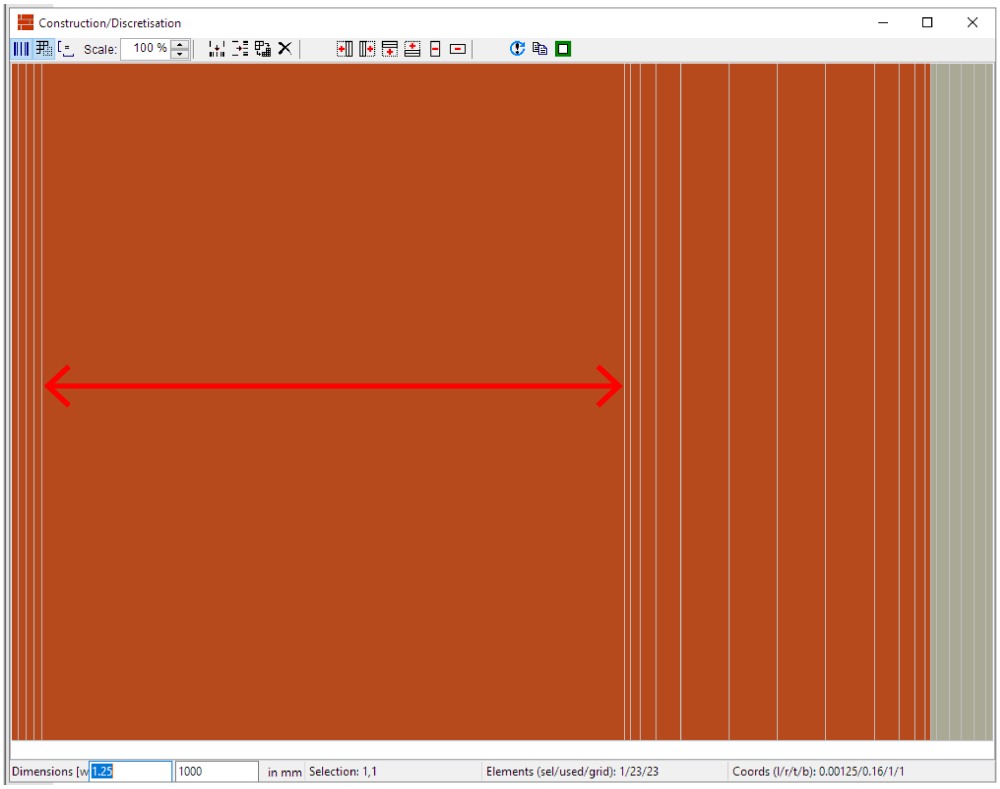

Figure 2: Part of the masonry wall component is not discretised. The dimension of this column will be edited by SHARK, after which the column is automatically discretised.

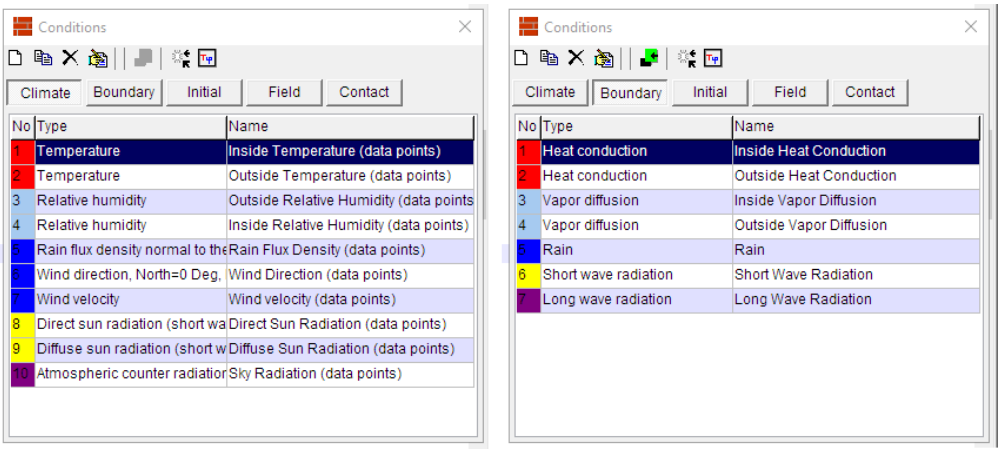

Figure 3: Example of how to name the interior and exterior climate and boundary conditions.

2.4 Assigning the outputs

The outputs are assigned as usual. No formatting rules apply for the names of the output files. For clarity, however, it is recommended not to include any numbering in the file names, as SHARK will add a numbering (based on the sample number) as well.

Note: Assign only the outputs you really need for your project, as writing the output during the simulations is resource-intensive (even when you have a solid state drive). In our experience, the bottleneck for machines with more than 8 CPU's are not the actual computations, but writing the output to the disk. Hence, reducing the amount of output might reduce the total simulation time.

3 Creating a sampling scheme and generating variation DELPHIN project files

This step is performed by the script `config.py`. Before SHARK can do its work, you need to define the parameter distributions and all the other variables needed for the probabilistic assessment.
First, you need to define the interior climate model and the scenario layer, as shown in Listing 1. The scenario layer always defines the interior climate, based on the selected model. Currently, two European Standards are implemented: EN 15026 [2007] and EN 13788 [2012]. Table 1 shows which values should be entered for which model. Note that it is possible to enter only one value or only some of the possible values (e.g. for EN 13788 `[2,3]` and for EN 15026 `['a']` are both valid entries). It is not possible to leave out the scenario layer, you should always select at least one scenario. It is also not possible to combine the two interior climate models in the same project.

```
# Interior climate model
interior_climate_type = 'EN15026' # Use 'EN15026' or 'EN13788'
# Scenario layer - scenario's
scenario = {'parameter':'int climate', 'value':['a','b']} # currently, only one scenario
    parameter is supported
```

Listing 1: Define the interior climate model and scenario layer.

Table 1: The possible entries for `value` in the DataFrame `senario` for the available interior climate models.

model	`value`
EN 15026	`['a', 'b']` or `[1, 2]`
EN 13788	`[1, 2, 3, 4]`

Next, the parameter distributions of the uncertainty layer need to be defined, as shown in Listing 2. The distributions are combined in a DataFrame with three columns: `parameter`, `type` and `value`. Table 2 shows the possible values for `parameter` and their meaning, while Table 3 shows the possible distribution types and their respective value formatting. The parameter value `'ext climate'` should always be defined, all other parameters are optional. If you want to change the material of a building component, the parameter name should consist of the word `material` and the material name you gave to it in DELPHIN. In the example, the brick type of the masonry wall will be changed, hence the materials are named `Brick1`, `Brick2` and `Brick3` (`Brick1` is assigned to the geometry) and the parameter value is `brick material` (see line 11-13 of Listing 2). The command is case-insensitive and the word order does not matter. If you want to change the the dimensions of a building component, you do not need to add the material name (see further how SHARK handles this). However, this is recommended for clarity.

```
# Uncertainty layer - Parameter distributions
distributions = pd.DataFrame([{'parameter':'solar absorption',
                               'type':'uniform',
                              'value':[0.4,0.8]},
                             {'parameter':'ext climate',
                               'type':'discrete',
                              'value': ['Oostende','Gent','StHubert','Gaasbeek']},
                             {'parameter':'exterior heat transfer coefficient slope',
                               'type':'uniform',
                              'value':[1,4]},
                             {'parameter':'brick material',
                               'type':'discrete',
                              'value':['Brick1','Brick2','Brick3']},
                             {'parameter':'scale factor catch ratio',
                               'type':'uniform',
                              'value':[0,2]},
                             {'parameter':'wall orientation',
                               'type':'uniform',
                              'value':[0,360]},
                             {'parameter':'start year',
                               'type':'discrete',
                              'value':24},
                             {'parameter':'brick dimension',
                               'type':'uniform',
                              'value':[0.2,0.5]},
                             ])
```

Listing 2: Define the parameter distributions.

Next, you need to define of which building component you want to change the dimensions, as shown in Listing 3. The variable `buildcomp` is a list with two elements: the first element is the first column or row of the building component in DELPHIN; the second element is the last column or row of the building component. You can find the column/row numbering in DELPHIN by selecting a cell; at the bottom of the Construction window, you will see *Selection: x,y*: x is the column, y is the row. The variable `buildcomp_elem` defines which column/row need to be changed in dimension (this column/row should not be discretised!). The variable `dir_cr` tells SHARK whether it is a column (`'column'`) or a row (`'row'`).

```
# Change dimensions of building component
buildcomp = [1,17]
buildcomp_elem = 5
dir_cr = 'column'
```

Listing 3: Define the building component of which the dimensions need to be changed.

If you don't want to change the dimensions of a building component, you can replace Listing 3 by:

```
# Change dimensions of building component
buildcomp = None
buildcomp_elem = None
dir_cr = None
```

Next, you need to define some parameters about the exterior climate, as shown in Listing 5. You have mainly two possibilities for the exterior climate: you use yearly cyclic data (e.g. the climate data provided by DELPHIN), or you use climate data that consists of multiple years (e.g. the Climate for Culture data). In the latter case, you need to define how many years the simulation runs via the variable `number_of_years`. Note that this should be a whole number, even if your simulation does not run for full years. In the example the simulation starts in September and runs for 6 years and four months. Hence, `number_of_years` is 7 as 7 years of data will be used. Furthermore, you need to define whether you want to use wind driven rain via the variable `simulate_wdrain`. If this is the case, it is important that you used the climate condition type *Rain flux density normal to the wall surface* and the boundary condition type *Rain* of kind *Imposed flux*.

Table 2: The possible entries for `parameter` in the DataFrame `distributions`.

Parameter	**Meaning**
`wall orientation`	the orientation of the wall
`wall inclination`	the inclination of the wall
`exterior heat transfer coefficient`	the heat transfer coefficient, defined by the exterior boundary condition, type 'heat conduction', kind 'exchange coefficient'
`exterior heat transfer coefficient slope`	the heat transfer coefficient slope, defined by the exterior boundary condition, type 'heat conduction', kind 'boundary layer'
`interior vapour diffusion transfer coefficient`	the vapour diffusion transfer coefficient, defined by the interior boundary condition, type 'vapour diffusion', kind 'exchange coefficient'
`exterior vapour diffusion transfer coefficient`	the vapour diffusion transfer coefficient, defined by the exterior boundary condition, type 'vapour diffusion', kind 'exchange coefficient'
`exterior vapour diffusion transfer coefficient slope`	the vapour diffusion transfer coefficient slope, defined by the exterior boundary condition, type 'heat conduction', kind 'boundary layer'
`solar absorption`	the solar absorption, defined by the exterior boundary condition, type 'short wave radiation', kind 'direct sun radiation model'
`scale factor catch ratio`	the rain exposure coefficient, defined by the exterior boundary condition, type 'rain', kind 'imposed flux'
`ext climate`	the exterior climate location
`start year`	the start year for the exterior climate (only meaningful if using climate data with more than one year
∗ `material`	the material that needs to be changed
∗ `dimension`	the building component of which the dimension need to be changed

Replace * by material name, e.g. 'brick'

Table 3: The possible entries for `value` in the DataFrame `distributions`.

Distribution type	**Distribution parameter values**
`uniform`	[*minimum value*, *maximum value*]
`normal`	[*mean*, *standard deviation*]
`discrete`	[*value 1*, *value 2*, *value 3*, ...] or *maximum value*

SHARK has it's own rain model, based on Blocken [2004], which is more accurate then the rain model of DELPHIN. SHARK will automatically calculate the rain flux based on the provided exterior climates and the wall orientation. If you do not wish to include wind-driven rain, set `simulate_wdrain` to `False`. This is usually not recommended, although it can be useful in some cases (e.g. in case of a wall with façade cladding).

```
# Climate info
number_of_years = 7
simulate_wdrain = True
```

Listing 4: Provide the exterior climate details.

Finally, you need to provide SHARK with the details about the sampling scheme, as shown in Listing 4. Currently, there are two possibilities for `sampling_strategy`:

1. `'load'`: an existing raw sampling design (values not yet converted to parameter distributions) is loaded from the folder `Sampling scheme`. SHARK converts the raw sampling design to a sampling scheme. The variables `samples_per_set` and `sets` are ignored, you can set both to `Null`.
2. `'sobol'`: a new sampling design is created, using the scrambled sobol sampling strategy. Subsequently, this sampling design is converted to a sampling scheme. The variables `samples_per_set` and `sets` are required.

The final result is the same: a sampling scheme that will be used for the probabilistic assessment. If you don't have an existing raw sampling design for your project, use `'sobol'`. The variable `samples_per_set` defines how many samples are created per set. This number should be a multiple of all considered discrete input parameter values. In this example, this is 4 exterior climates, 3 brick types and 24 year; hence, 24 is the least common multiple. The variable `sets` defines how many sets are created. Because in this example two scenario's were defined in the scenario layer. The total number of samples will be *number of scenario's* x `samples_per_set` x `sets`. For this example, this is 2 x 24 x 3 = 144 samples.

```
# Sampling details
sampling_strategy = 'sobol'
samples_per_set = 18
sets = 3
```

Listing 5: Enter the sampling scheme details.

The remainder of the script `config.py` should not be edited, unless you know what you are doing.

4 Running the variation DELPHIN project files

This step is performed by the script `run.py`. Here you only need to provide limited information to SHARK, as shown in Listing 6. The variable `check_finished_simulations` allows you to skip the files that were already simulated successfully. If `check_finished_simulations` is set to `True`, the solver checks which files were already simulated successfully and skips those. If you run the simulations for the first time, set `check_finished_simulations` to `False`, to save time. The variable `cpu_unused` allows you to specify how many CPU's SHARK can not use. On machines with only 4 cores, it is recommended to set this variable to `1` to prevent the machine from freezing. On machines with 8 or more cores, you could set this variable to `0`. Finally, you need to specify the path to the DELPHIN solver via the variable `delphin_executable`. If this path is not correct, SHARK will not be able to call the solver and the simulations will not run. On windows machines, this path is usually the one used in the example and thus should not be changed. The remainder of the script `run.py` most not be changed.

```
# Chech first which simulations were already run successfully? and skip these
check_finished_simulations = True
# How many CP's schould NOT be used?
cpu_unused = 0
# Specify the path to the Delphin executable
delphin_executable = 'C:\Program Files (x86)\IBK\Delphin 5.8\delphin_solver.exe'
```

Listing 6: `run.py`.

5 Post-processing the output of the DELPHIN simulations

The script `postprocess.py` is an example of how to use the dpm module of SHARK for post-processing DELPHIN output with damage prediction models. The output of the DELPHIN simulations is read into Python and subsequently processed by the desired damage prediction model. Currently, three damage prediction models are implemented in the dpm module of SHARK:

- Evaluate frost damage by the number of moist freeze-thaw cycles. The freeze-thaw cycles are calculated by the TUDresden equation, not by the DELPHIN ice model. A 'moist' freeze-thaw cycle is a freeze-thaw cycle that occurs in combination with a moisture content in the brick (usually at 0.5 cm from the exterior surface) that is high enough to induce frost damage, defined by a moisture content higher than a certain percentage of the saturated moisture content.
- Evaluate mould growth risk by the Mould Index calculated by the Updated VTT model Viitanen et al. [2011]. The Mould Index is an indicator for the mould growth range and has level 0 (no growth) to 6 (Heavy and tight growth, coverage about 100 %).
- Evaluate wood decay risk by the wood mass loss calculated by the VTT wood decay model Viitanen et al. [2010]. The wood mass loss is an indicator for the wood decay rate and ranges from 0 to 100 %.

Table 4 gives an overview of the damage prediction models and the required output of DELPHIN.
Other damage prediction models exist, though they have not (yet) been implemented. Note that all damage prediction models are not very accurate and caution is required when using them. A comparison of the relative increase in risk (compared to a reference construction) might be more reliable than an absolute comparison of the damage risk indicators.
As post-processing is very project specific, the script `postprocess.py` will not be discussed in detail. The example is documented enough, as are all the functions of the dpm module.

Table 4: Overview of the damage prediction models and the required output of DELPHIN.

Damage pattern	**Prediction model**	**DELPHIN output**
Frost damage	Moist freeze-thaw cycles via TUDresden equation criterion	T, RH, moisture saturation degree
Mould growth	Updated VTT mould growth model	T, RH
Decay of wooden beam ends	VTT wood decay model	T, RH

6 Monitoring convergence

References

Bert Blocken. *Wind-driven rain on buildings measurements, numerical modelling and applications*. PhD thesis, KU Leuven, 2004.

European Committee for Standardization. En 15026: Hygrothermal performance of building components and building elements. assessment of moisture transfer by numerical simulation, 2007.

European Committee for Standardization. En 13788: Hygrothermal performance of building components and building elements - internal surface temperature to avoid critical surface humidity and interstitial condensation - calculation methods, 2012.

Liesje Van Gelder, Hans Janssen, and Staf Roels. Probabilistic design and analysis of building performances: Methodology and application example. *Energy and Buildings*, 79:202–211, 2014.

Hannu Viitanen, T. Toratti, L. Makkonen, R. Peuhkuri, Tuomo Ojanen, L. Ruokolainen, and J. Räisänen. Towards modelling of decay risk of wooden materials. *European Journal of Wood and Wood Products*, 68 (3):303–313, 2010.

Hannu Viitanen, Tuomo Ojanen, and R. Peuhkuri. Mould growth modelling to evaluate durability of materials. In *Proceedings of the 12DBMC - International Conference on Durability of Building Materials and Components*, pages 1–8, 2011.