Stochastic simulation of building envelope performance: methodology and implementation

Wang, Lin; Defo, Maurice; Lacasse, Michael A.

Publisher's version / Version de l'éditeur:
https://doi.org/10.4224/40003115
Client Report (National Research Council of Canada. Construction); no. A1-020287, 2022-06-21

NRC·CNRC CONSTRUCTION

Stochastic simulation of building envelope performance: methodology and implementation

Author(s): Lin Wang, Maurice Defo, Michael A. Lacasse
Report No.: NRCC-CONST-56604E
Report Date: June 21, 2022
Contract No.: A1-020287
Agreement date: August 19, 2021

National Research Council Canada
Conseil national de recherches Canada

Canada

Stochastic simulation of building envelope performance: methodology and implementation

Author

Wang, Lin Digitally signed by Wang, Lin Date: 2023.07.13 12:54:30 -04'00'

Lin Wang
Assistant Research Officer
Building Envelope and Materials,
NRC Construction Research Centre

Approved

Armstrong, Marianne Digitally signed by Armstrong, Marianne DN: cn=Armstrong, Marianne, c=CA, o=gc, ou=NRC-CNRC, email=marianne.armstrong@cnrc-nrc.gc.ca Date: 2023.07.13 15:09:29 -04'00'

Marianne Armstrong, RCO
Program Leader
Climate Resilient Build Environment (CRBE) Initiative
NRC Construction Research Centre

Report No: NRCC-CONST-56604E
Report Date: June 21, 2022
Contract No: A1-020287
Agreement date: August 19, 2021
Program: CONST CRBE Climate Resilient Built Environment (A32N)

(This page is intentionally left blank)

Table of Contents

List of Figures

List of Tables

Executive summary

It is well recognized that climate changes will have an impact on building performance in different aspects, such as the performance with respect to whole building energy consumption, to indoor thermal comfort as well as to the hygrothermal performance of building envelope. Climate-resilient building design has become crucial for constructing new buildings or rehabilitating existing buildings when adapting to the changing climate.

NRC Construction Research Centre has carried out a Climate-Resilient Built Environment (CRBE) project: "Decision Support Tools for Building Envelope Performance Assessment" (2021-2026), intended as means to develop decision support tools, including guides and models for the design of resilient new buildings, and the rehabilitation of existing buildings to ensure that existing and future climate loads and extreme weather events are considered. To achieve these goals, the NBC Part 9 walls will be investigated through hygrothermal simulation and stochastic simulation to take the uncertainties of climate change into consideration.

In the CRBCPI project carried in the past five years (2016-2021), an extensive set of hygrothermal simulations were carried out to evaluate the mould growth performance of different types of wood-frame building envelopes under historical and future climatic conditions across a number of major cities in Canada. The hygrothermal simulations were performed by following the approach described in *Guideline on Design for Durability of Building Envelopes*[1], and the simulation results were presented in the report *Results from Hygrothermal Simulations and on Durability and Resilience of Wall Assemblies to Climate Change*[2]. To consider the uncertainties of the input parameters, such as the material properties, rain water penetration moisture loads and cladding ventilation rates, stochastic simulations need to be performed in the CRBE project (2021-2026). The stochastic simulation procedure should be generalized, and the generalized procedure can be applied to the CRBE project related to hygrothermal performance analysis and moisture damage risk assessment of building envelopes.

The information provided in this report describes the process of rationalisation of the procedure used for hygrothermal performance analysis and probabilistic risk assessment of moisture related damage of building envelopes using stochastic simulation. It is intended for use by expert practitioners who have knowledge of hygrothermal simulation, and require hygrothermal performance results to assess climate resilient building envelope design, considering the uncertainties in input parameters. A complete stochastic simulation procedure requires the following steps:

- The quantification of stochastic variables
- Applying proper sampling techniques to generate stochastic models
- Implementing stochastic simulations
- Data visualization of stochastic results

Generally, the stochastic simulation needs to be implemented in a third-party programming environment, in which a hygrothermal simulation engine can be called, and the stochastic models with randomly assigned variables can be repeatedly launched. In this report, the hygrothermal simulations were executed using Delphin 5.9.6, a commercial hygrothermal simulation program. Python 3.4 software was used for generating probability distributions of stochastic variables, implementing advanced sampling techniques to generate stochastic models, launching stochastic simulations and visualizing the stochastic results. A typical 2x6-inch wood-frame wall was used to demonstrate the stochastic simulation procedure; the stochastic simulations were performed under the historical and future climatic conditions of Ottawa.

Although the stochastic simulation framework was developed for the purpose of hygrothermal performance analysis and moisture damage risk assessment of building envelopes, this procedure can also be adapted for summer-time overheating risk assessment.

[1] Lacasse, M. A., Ge, H., Hegel, M., Jutras, R., Laouadi, A., Sturgeon, G., & Wells, J. (2018). *Guideline on Design for Durabiliry of Building Envelopes.* Ottawa: National Research Council of Canada.

[2] Defo, M., Wang, L., & Lacasse, M.A. (2021). Results from Hygrothermal Simulations and on Durability and Resilience of Wall Assemblies to Climate Change. Ottawa: National Research Council of Canada.

Stochastic simulation of building envelope performance: methodology and implementation

1 Introduction

Global warming will inevitably occur in the future, with more frequent extreme weather events, such as heavy rainfall events and heat waves (IPCC, 2021). It is projected that the annual average temperature across Canada will increase by 6.3℃ for a high emission scenario (RCP 8.5) by the end of the 21st century as compared to the reference period of 1986-2005, and there will be a greater amount of precipitation in all parts of Canada in the future (Zhang et al., 2019). The changing climate may influence building performance in different aspects, such as building energy consumption performance, the indoor thermal comfort performance as well as the hygrothermal performance of the building envelope.

To investigate the impact of climate change on building performance, the NRC Construction Research Centre has undertaken a Climate-Resilient Buildings and Core Public Infrastructure (CRBCPI) Program, which is intended as a means to develop decision support tools, including codes, guides and models for the design of resilient new, and the rehabilitation of existing buildings and CPI in key sectors to ensure that existing and future climate loads and extreme weather events are considered. In the CRBCPI Program, two aspects of building performance were studied: (i) the hygrothermal response of building envelopes to a changing climate, and; (ii) the overheating of buildings due to urban heat island effects as may arise due to climate change.

For the hygrothermal response study, an extensive set of hygrothermal simulations were carried out to evaluate the mould growth performance of different types of wood-frame building envelope under historical and future climatic conditions across a number of major cities in Canada. The hygrothermal simulations were performed by following the approach described in Lacasse et al. (2018) and the simulation results were presented in the report by Defo et al. (2021). Although hygrothermal modelling is a powerful tool that can be used for heat and moisture performance analysis and moisture related damage prediction of building envelopes, the commonly used hygrothermal modelling programs require deterministic input parameters to perform the hygrothermal simulation, which may not be able to capture the moisture damage risks caused by the stochastic nature of the input parameters. Stochastic simulation permits practitioners to take into consideration the randomness in values of the input parameters. Instead of using deterministic values for the input parameters, for stochastic simulation, random numbers are used that are taken from a probability distribution of each parameter. A proper sampling technique is applied for the selection of a set of stochastic parameters that are then used for generating stochastic models. Thereafter, stochastic simulations can be performed using the stochastic models to generate a set of stochastic results, from which moisture damage risks can be assessed as based on the results of the stochastic simulations.

To consider the uncertainties of the input parameters, such as the material properties, rain water penetration moisture loads, and cladding ventilation rates, stochastic simulations were performed for selected cities and different types of walls. Since the uncertainties of the input parameters exist universally, the stochastic simulation procedure, as applied for this hygrothermal simulation task, be generalized, and thus, be applied to the CRBE project (2021-2026) and other research projects related to hygrothermal performance analysis and moisture damage risk assessment of building envelopes.

The information provided in this report describes the process of rationalisation of the procedure used for hygrothermal performance analysis and probabilistic risk assessment of moisture related damages in building envelopes using stochastic simulation. The stochastic simulation framework can be used by expert practitioners who have knowledge of hygrothermal simulation and require hygrothermal performance results to assess climate resilient building envelope design. The stochastic simulation procedure developed in this report was essentially based on a randomized Sobol sequence based sampling method, one of the Randomized Quasi-Monte Carlo (RQMC) sampling methods, which has a high sampling efficiency and allows a conservative error estimation of the stochastic results. The stochastic simulation procedure was elaborated step by step with the aid of executable Phyton 3.4 software scripts.The scripts were designed for use with Delphin 5.9.6, a widely used hygrothermal

simulation program. To demonstrate the application of the stochastic simulation procedure, a typical 2x6-inch wood-frame wall under historical and future climatic conditions of Ottawa was used as a case study. The mould growth risk was assessed through the use of stochastic simulation, and the effectiveness of different possible risk mitigation strategies was evaluated.

2 Methods

Generally, a complete stochastic simulation procedure requires the following steps:

- Create a base model
- Quantify the stochastic variables
- Apply proper sampling technique to generate stochastic models
- Implement stochastic simulation
- Visualize stochastic data results

Since most of the hygrothermal / energy simulation programs do not integrate a stochastic simulation module, the stochastic simulation needs to be implemented using a third-party programming environment, in which a hygrothermal / energy simulation engine can be called, and the stochastic models, having randomly assigned variables, can be launched repeatedly. In this report, the creation of a base case model and hygrothermal simulation were implemented using Delphin 5.9.6 software, a widely used hygrothermal simulation program. The Python 3.4 software was used for implementing those tasks related to the stochastic procedure, such as generating probability distributions of stochastic variables, implementing advanced sampling techniques to generate stochastic models, launching stochastic simulations and visualizing the stochastic results. Figure 1 shows the overall operating procedure between Delphin and Pyhon.

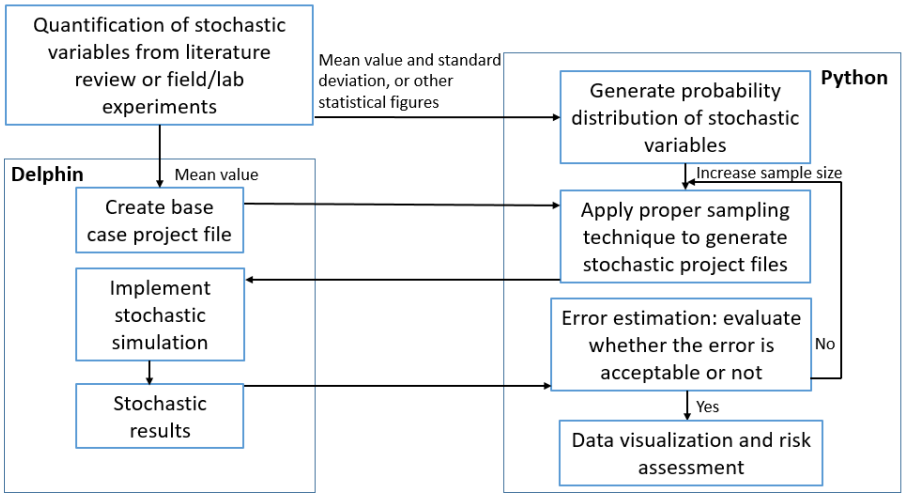


Figure 1 Operating procedure between Delphin and Python

In the following sections, focus will be on elaborating the steps implemented in Python, with the aid of scripts written in Python 3.4 software language. These steps can also be realized using other programming languages.

2.1 Quantification of stochastic variables

The goal of quantifying a stochastic variable is to generate the probability distributions of those stochastic variables from which random numbers can be sampled based on advanced sampling techniques. The most commonly used probability distributions for stochastic variables found in a hygrothermal simulation are the uniform distribution and normal distribution. The type of probability distribution of a specific stochastic variable should be the one that best fits the values obtained from field or lab measurements. For example, the values of

cladding ventilation rate (ACH) from field measurements for a one year period tend to follow a normal distribution; as such, the stochastic variable for ACH can be considered to follow a normal distribution. Whereas, for the rain deposition factor, it could be assumed to follow a uniform distribution. For a specific roof design, the variation in rain deposition factor is dependent on different building shapes and the local suroundings, which equally occur in the building stock. To identify a uniform distribution, the maximum and minimum values of the stochastic variables are required; for a normal distribution, the mean value and standard deviation are required. Knowing these statistical figures, the random numbers can be generated from a specific probability distribution using the following scripts in Python:

```
import numpy as np

# The script to generate random numbers from a uniform distribution
# a - lower limit
# b - upper limit
x_uniform = np.random.uniform (a, b)

# The script to generate random numbers from a normal distribution
# mu - mean value
# sigma - standard deviation
x_norm = np.random.normal(mu, sigma)
```

Theoretically, the normal distribution has no upper or lower limits, however, some input parameters for the hygrothermal simulation have to be limited to a certain range to ensure a reasonable physical meaning of the parameter value. Therefore, the random number generated from the normal distribution should be truncated within a certain range. For example, from field measurements undertaken from a building located in the coastal climatic conditions of Vancouver, it was shown that the cladding ventilation rate (ACH) for a brick veneer cladding wall having a 25-mm drainage cavity and 2 openings at the bottom and top of the wall (ventilated design) was in the range between 1 and 11, with a mean value of 6 (Simpson, 2010). In this instance, the random number range for ACH generated from the normal distribution should be limited to between 1 and 11. The standard deviation can be estimated based on the range of the stochastic variable and the mean value using equation (1):

$$Std = \frac{\max - min}{6} \qquad (1)$$

Where:

Std – standard deviation

Max- upper limit of the random numbers

Min- lower limit of the random numbers

The dataset with the standard deviation calculated from equation (1) will have approximately 99% of the data within the specified maximum and minimum values. The truncated random numbers from a specific normal distribution can be generated using the following python scripts:

```
import scipy.stats as stats

# Define lower and upper limits
lower, upper = 1, 11

# Define mean value and standard deviation
mu, sigma = 6, 2

# Generate the normal distribution with defined mean value ad standard deviation
# as well as the lower and upper limits
ach_truncated = stats.truncnorm((lower - mu)/ sigma, (upper - mu)/ sigma, loc = mu, scale = sigma)

# Generate the random numbers with specified sample size, the sample size in this case is 10000
ach_truncated_samples =  ach_truncated.rvs(10000)
```

Figure 2 shows the histogram of the stochastic ACH.

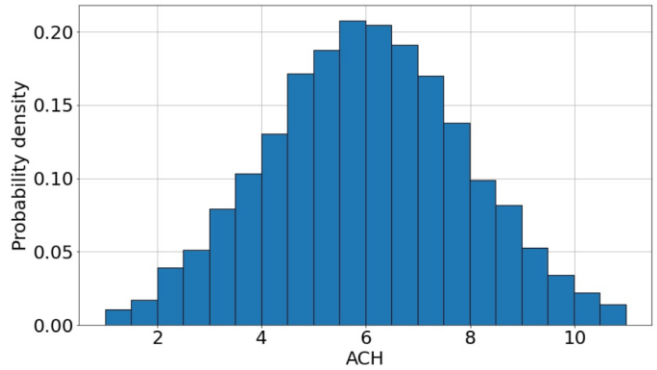


Figure 2 Histogram of ACH of a typical brick veneer wall with two openings at the top and bottom

Sometimes, one input parameter is, to a degree, correlated to another parameter. For example, the water absorption coefficient (A-value) of brick is correlated with the saturation water content with a correlation coefficient of 0.6 (Zhao et al., 2015). In this case, the generated random numbers of these two parameters should reflect this correlation. The random numbers from two correlated normal distributions can be generated using the following Python scripts:

```
import scipy.stats as stats
numpy.random.multivariate_normal (mean, cov, size)
```

Where: mean - an array of the mean values

cov - the covariance matrix

size - the sample size

As an example, the stochastic A-value and saturation water content were generated with a correlation coefficient of 0.6 using following Python scripts:

```
# Create array containing the mean values of A-value  and saturation water content
np_Brick_properties_mean =  np.array([0.0268, 216.7])

# Create matrix containing the standard deviations of A-value and saturation water content
np_Brick_properties_std = np.array([[0.0035, 0], [0, 29]])

# Create matrix containing correlation coefficients
np_Brick_properties_corr = np.array([[1, 0.6], [0.6, 1]])

# Calculate covariance matrix
np_Brick_properties_cov = np.matmul(np_Brick_properties_std, np_Brick_properties_corr )
np_Brick_properties_cov = np.matmul(np_Brick_properties_cov, np_Brick_properties_std )

# Generate correlated random numbers with specified sample size
np_Brick_properties = np.random.multivariate_normal(np_Brick_properties_mean, np_Brick_properties_cov, 10000)
```

Additional details of the Python software scripts for the random number generation of different probability distributions are provided in Appendix A.

2.2 Implementation of sampling

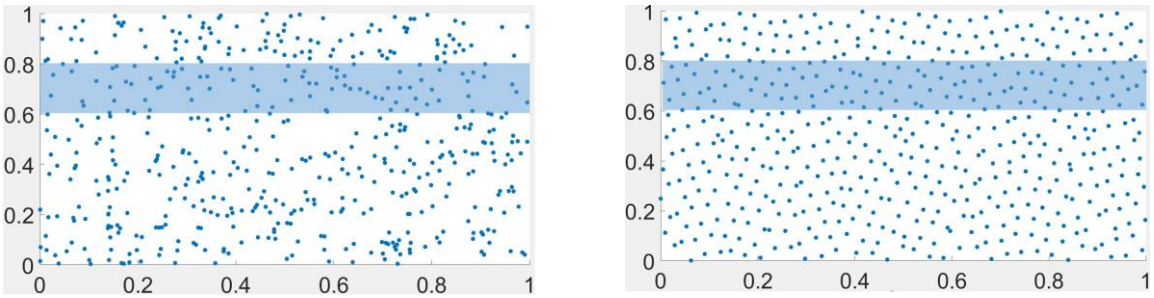
The sampling should be based on the random numbers from the probability / cumulative distributions of the stochastic variables. There are different sampling techniques available in the literature. A detailed review of the application of different sampling techniques on stochastic hygrothermal simulation can be found in Wang et al. (2021). The Sobol sequences belong to the family of quasi-random sequences that are designed to generate samples from multiple parameters as uniform as possible over multi-dimensional space (Saltelli et al., 2010). Figure 3 shows the comparison between random sampling and Sobol sequence based sampling in space of [0, 1]. It can be seen that the points sampled based on the Sobol sequence (Fig. 3b) are more evenly distributed in a two dimensional space than those sampled based on random sampling (Fig. 3a). For any specific interval, for example the interval of (0.6, 0.8) of the y-axis variable, the points generated using the Sobol sequence have fewer gaps and clusters than those generated using a random sampling method. Therefore, when the expectations and standard deviations of the stochastic results are calculated at that interval and with the same sample size, the results from Sobol sequence based sampling are less likely to deviate from the true values than those from the random sampling.

The implementation of the Sobol sequences based sampling should be implemented in three steps:

1. Generate the random numbers from the probability / cumulative distributions of the stochastic variables, (step illustrated in section 2.1)

2. Generate the Sobol points from the Sobol sequences (illustrated in section 2.2.1).

3. The sampling of the random numbers for stochastic variables based on Sobol points (illustrated in section 2.2.2).



a) Random sampling b) Sobol sequence based sampling

Figure 3 Sampled points from two sampling methods

2.2.1 Sobol point generation

To implement a Sobol sequence based sampling in Python, one needs to install the module “PyTorch”[3]. For different installation environments, the installation command can be found in PyTorch (2021).

The Sobol points that are used for sampling the stochastic variables need to be generated based on Sobol sequences, which belong to the family of low-discrepancy sequences. The points extracted from low-discrepancy sequences are capable of filling the hyperspace $[0,1]^s$ by minimizing the gaps (Wikipedia, 2021). The Sobol sequences are deterministic low-discrepancy sequences. This means that the points generated by Sobol sequences are fixed in the sample space, and the conventional error estimation methods based on the central limit theory are not valid for the Sobol points (Hou et al., 2017).To evaluate the error in stochastic results, the

[3] https://download.pytorch.org/whl/torch_stable.html

Sobol sequences need to be randomized. Thereafter, different sets of randomized Sobol points can be generated from the different randomized Sobol sequences. Several algorithms can be used to randomize the low-discrepancy sequences (L'Ecuyer and Lernieux, 2002). It has been shown that the scrambling method is suitable for randomizing Sobol sequences (Hou et al., 2017), and the scrambling method can be readily implemented using the module found in PyTorch (2021). Therefore, in this study, the Sobol sequences were randomized using the scrambling method.The scripts for generating Sobol points are provided in Appendix B.

In principle, the dimension of the Sobol sequences should be equal with the number of stochastic variables. However, if there are two correlated stochastic variables, the two correlated variables need to be sampled by only one column of the Sobol sequence to guarantee their correlation. The number of Sobol points can be very large, (e.g. 100,000) which helps guarantee the maximum sample size required for stochastic simulation. When sampling the data from the probability distributions based on the Sobol points, one could start with small sample size, then sample the stochastic variables, run the simulation, and check the standard error. If the standard error is not acceptable, then the Sobol points need to be added, and the sampling and simulation need to be repeated until the standard error reaches an acceptable level (see Figure 1).

2.2.2 Sample stochastic variables based on Sobol points

After the generation of random numbers from different probability distributions and the randomized Sobol points, the stochastic variables can be sampled based on the Sobol points. Since each stochastic variable has a cumulative distribution, the random numbers can be selected based on their percentiles in their cumulative distribution, and the percentiles of the selected random numbers should be the same value as the Sobol points.. The sampling can be implemented through following the Python scripts:

```
import numpy as np
for i in range (n):
    np_Sobol_stochastic_variable [n] =  np.percentile (np_Random_number, 100 * np_Sobol_point[n])
```

Where:

n- sample size of random numbers selected based on Sobol points

np_Sobol_stochastic_variable – numpy array of sampled random numbers based on the original random numbers and the Sobol points

np_Random_number – numpy array of original random numbers generated from a specific probability distribution

np_Sobol_point – numpy array of Sobol points

What should be noted is that the size of the original random numbers and Sobol points should be much higher than the sampled random numbers. For example, the original random numbers and Sobol points could have a very large size (100,000), therefore, the size of sampled random numbers could range from hundreds to thousands, and the probability of repeated sampling could then be reduced.

When a stochastic variable has different categories, i.e. different environmental conditions or design options, the Sobol points should be divided into different sub-spaces. As such, the sampling can be implemented within each sub-space. For example, if a stochastic variable can be categorized into three categories, then the Sobol points should be divided into three sub-spaces [0,1/3],[1/3,2/3] and [2/3,1]. When the value of the Sobol point is lower than 1/3, the stochastic Sobol variable should be sampled within the first interval. Similarly, when the value of Sobol point lies between 1/3 and 2/3, the sampling will be implemented within [1/3, 2/3], and so on. The following Python scripts can be used to sample the stochastic variables having different categories:

```
for i in range (n):
    if np_Sobol_point[i] < 1/3
       np_Sobol_stochastic_variable [i] =  np.percentile (np_Random_number, 100 * (np_Sobol_point[i]-0)/(1/3-0))
    elif np_Sobol_pint[i] >=1/3 and np_Sobol_pint[i] <2/3
       np_Sobol_stochastic_variable [i] =  np.percentile (np_Random_number, 100 * (np_Sobol_point[i]-1/3)/(2/3-1/3))
    elif np_Sobol_pint[i] >=2/3 and np_Sobol_pint[i] <1
       np_Sobol_stochastic_variable [i] =  np.percentile (np_Random_number, 100 * (np_Sobol_point[i]-2/3)/(1-2/3))
```

Where:

n- sample size of random numbers selected based on Sobol points

np_Sobol_stochastic_variable – a numpy array of sampled random numbers

np_Random_number – a numpy array of original random numbers generated from a specific probability distribution

np_Sobol_point – a numpy array of Sobol points

For discrete variables, a similar approach is applied to sample the random values. For example, for the climate data set is comprised of 15 climatic realizations for a specific climatological period and for a specific city as such, the space of Sobol points can be divided into 15 intervals. When the Sobol point falls into the first interval, then the R01 is selected, when the Sobol point falls into the second interval, then the R02 is selected and so on. The detailed Python scripts can be found in Appendix C.

The sampled random numbers should be saved in a csv file, as the use of such a file type permits ease of re-loading when generating the stochastic project files (dpj files) for use in DELPHIN.. For discrete stochastic and continuous variables having different categories, the categories of each sampled random number should be recorded and saved to a csv file, which will make it easier for differentiating the stochastic results in different categories. What should be noted is that, since there are different randomized sets of sampled random numbers, there will be different csv files. Each csv file include one set of randomized stochastic variables sampled based on one set of the Sobol sequences.

2.3 Generate stochastic DELPHIN project files (dpj files)

The generation of stochastic DELPHIN project files (dpj files) is based on the csv files in which are recorded sampled random numbers. The general procedure for generating stochastic project files is the following:

1. Load the base case for the DELPHIN project file

2. Load the random numbers sampled based on Sobol sequences

3. Identify the positions of the stochastic variables (original values) in the dpj file, all the positions of the stochastic variables were identified simultaneously

4. Replace the original values in the base case file with the random values

5. Save the new stochastic DELPHIN project file

There are different sets of randomized numbers, therefore the stochastic dpj files should be saved in different folders depending on the different sets of randomized numbers. The stochastic variables sampled by Sobol points can be categorized into stochastic material properties, boundary conditions (rain deposition factor), heat/moisture generation sources (ACH and moisture source due to rain leakage), and climate data. Different types of variables differ in the way they may vary from original values in the dpj files, since these variables have different formats in the original dpj file. The details in respect to varying different types of stochastic variables are elaborated in the following sections.

2.3.1 Material properties

The basic material properties, such as the saturation water content and the A-value, are used for stochastic sampling. Then, the basic stochastic properties are used to scale the material property curves such as the moisture storage function and liquid diffusivity. For the moisture storage function, the stochastic material property curves can be obtained by multiplying a stochastic coefficient as shown in equation (2):

Coefficient_stochastic = θ_{eff}_rand/ θ_{eff}_base (2)

Where:

θ_{eff}_rand – the random number of effective saturation water content sampled based on the sobol sequences

θ_{eff}_base – the value of effective saturation water content of the base model

For liquid diffusivity, the liquid diffusivity at the saturation water content can be calculated using equation (3), and thereafter, the stochastic coefficient can be calculated with equation (4)

Dl_{eff}_rand = (A-value_rand / θ_{eff}_rand)^2 (3)

Coefficient_stochastic = Dl_{eff}_rand / Dl_{eff}_base (4)

Where

Dl_{eff}_rand – the random number of liquid diffusivity at a specific value of random water absorption coefficient and random effective saturation water content

Dl_{eff}_base – the liquid diffusivity at the A-value and effective saturation water content of the base case

To vary the material properties, firstly, the basic material properties should be varied, then, the material property curves should be varied by means of the equations (2) and (4). Two Python functions were designed to modify the material properties in the dpj file. The detailed scripts to vary material properties are provided in Appendix D.

2.3.2 Boundary conditions and heat/moisture sources

The boundary conditions of the hygrothermal model include: heat/vapour exchange coefficients, rain deposition/exposure factors, short wave absorptivity and long wave emissivity. The heat/moisture source include: cladding ventilation rate (ACH) and, moisture source from rain leakage to wall assembly. For some of the parameters, the value can be directly varied within the dpj file; for example, the ACH and short wave absorptivity parameters. The following scripts provide an example of varying such parameters:

```
#   Extract base ACH
ACH_base = 'CONSTVALUE                = 5 1/h'
ACH_Indice = [j for j, s in enumerate(ProjectContent_New) if ACH_base in s]
```

```
# Vary ACH
ProjectContent_New[ACH_Indice[0]] = ACH_base.replace('5' , str(pd_Stochastic_Variables.iloc[i,3]))
```

Where:

ACH_base – the line of code of ACH in original dpj file

ACH_indice – the position of the line of code of ACH in ProjectContent_New

ProjectContent_New – a list to store the all the lines of code of newly generated stochastic project file

list_Stochastic_Variables – a list to store all the random numbers of different stochastic variales

For other boundary conditions such as the rain deposition/exposure factor, these are related to the ASHRAE wind-driven rain model, which in DELPHIN is not a built-in model to calculate wind-driven rain. Therefore, these two parameters cannot be varied by simply changing an input parameter. The procedure for varying the rain deposition/exposure factor is elaborated in Section 2.3.3. Similarly, the moisture source from rain leakage also relates to the wind-driven rain calculation, and as such, it is also described in Section 2.3.3.

2.3.3 Climate data

Wen performing meteorological modelling, the uncertainties in climate data that exist in different climatic realizations depend on different assumed initial conditions. Gaur et al. (2019) extracted 15 climatic realizations from the Canadian Regional Climate Model (CanRCM4), and performed bias correction on the extracted climate data set. Therefore, the uncertainties amongst the 15 climatic realizations need to be considered in stochastic simulation.

The climate data are incorporated in the climate data files (ccd files) as hourly time series. Therefore, given the variation in the different climatic realization, there is a need to simultaneously modify multiple time series. Accordingly, a Python function was designed to vary the climate data and generate the ccd files for use in DELPHIN simulations, this is given here:

```
def generating_ccd (ClimateData, list_Years, Wall_Orientation, F_D_E, RL_Percentage, RL_Thickness):
```

Where:

ClimateData – A list to store the climate data from original csv file of climate data

List_Years – A list of the years that climate data corresponding to

Wall_Orientation – the wall orientation

F_D_E – the product of rain deposition factor and exposure factor

RL_Percentage – moisture source of rain leakage (% of wind-driven rain)

RL_Thickness – the thickness of the element that receives the moisture source of rain leakage

The details of this function are provided in Appendix E. This function can be integrated into a loop to permit generating the stochastic ccd files with randomly varied climate data, and that repeatedly vary the climate data sets, wall orientation, rain deposition/exposure factor as well as moisture source for rain leakage. An example of generating the stochastic ccd file is also provided in Appendix E. What should be noted is that the folder structure of the ccd files should follow the structure presented in Figure 4. It can be seen that there are different sets of randomized stochastic climate data. For each set, there are different sets of ccd files, and these ccd file sets were sampled based on Sobol sequences. In each set of ccd files, there are different ccd files containing different time series of climate parameters. By using this structure, the ccd files in any randomly selected climate data set can be accessed according to their position in the Sobol sequence.

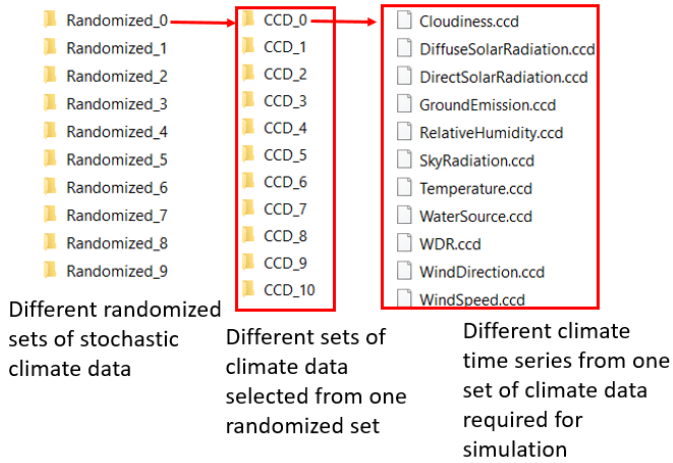


Figure 4 Folder structure of the stochastic climate files

2.3.4 Generate stochastic dpj files and launch stochastic simulation

When implementing the process of varying material properties, boundary conditions and heat/moisture source as well as the climate data, this should be completed in a loop in order to write all the generated random numbers into different newly generated dpj files; these can then be saved into specific locations. Appendix F shows the scripts for generating stochastic dpj files. What should be noted is that the folder structure for the stochastic dpj file should be the same as that of the ccd files. There should be different sets of randomized stochastic dpj files, each set of dpj files is saved in one folder. After the generation of stochastic dpj files, the stochastic simulations can be launched in parallel. Appendix G shows the scripts for launching the stochastic simulations.

2.4 Error estimation and data visualization

The error estimation of stochastic results should be based on a specific output, such as the average value of mould growth index or the moisture content over the simulation period. There are no standards or guidelines that regulate the error limit of stochastic simulations, however, the acceptable limit of error depends on the problem to be analyzed. For instance, the mould growth index has 6 levels, and each level represents one mould coverge rate. When the simulated mould growth index is in around 5, it might be reasonable to set an acceptable limit as 0.5, which results in a relative error in around 10%. However, in this report, the simulated mould growth index levels for the case study were between 0.5 and 2.0 depending on different climatic realizations and orientations. A limit of 0.5 will result in a relative error between 40% to 100%, which might be too large. As such, for the case study given in this report, an acceptable error limit of 0.1 was used, and this error results in a relative error between 5% and 20%. The standard error needs to be estimated at different sample sizes. The standard error of the mean value and standard deviation of the stochastic results can be calculated using equation (5) and equation (6) (Hou et al., 2019):

$$\bar{Q}_{n,r}(f) = \frac{1}{r}\sum_{i=1}^{r} Q_n^{(i)}(f) \qquad (5)$$

$$stderr(\bar{Q}_{n,r}) = \sqrt{\frac{1}{r(r-1)}\sum_{i=1}^{r}(Q_n^{(i)}(f) - \bar{Q}_{n,r}(f))^2} \qquad (6)$$

Where:

$Q_n^{(i)}(f)$ – the estimator of mean or standard deviation of the ith randomized sequence

$\bar{Q}_{n,r}(f)$ – the average of the estimators of r sets of randomized sequence, one set represents a specific realization of Sobol sequences

r – the number of randomized sequences

n – a specific sample size of the Sobol sequences, the total sample size is equal to r*n

f – the function that is used to calculate the performance indicator, in the case of mould growth performance, the mould growth indices are calculated through hygrothermal simulation

The scripts for estimating the standard error at different sample sizes, different categories of climatic scenarios, and orientations, as well as different design strategies are provided in Appendix H.

For data visualization, the stochastic results can be visualized by probability distribution, cumulative distribution and box plots. The overall procedure for data visualization is:

1. Extract the simulated results, such as relative humidity, temperature from the DELPHIN result folders.

2. Make the calculation for the performance indices, such as mould growth index, from the extracted DELPHIN results.

3. Make the plots of the stochastic results, such as the cumulative distribution, the probability distribution and box plot.

The plot can be made by the use of the matlibplot, seaborn or scipy modules. Appendix I shows an example of the cumulative distribution, probability distribution and box plot using seaborn.

3 Case study

3.1 Hygrothermal model

3.1.1 Wall configuration and material properties

Figure 5 shows the wall configuration of the wood-frame wall investigated in this report. The basic properties of the materials used in the wall components are given in Table 1. The material properties were taken from a series of NRC's material property reports (Kumaran et al., 2002; Maref et al., 2003; Kumaran et al., 2006). For the properties that were not found in NRC's reports, these were taken from Delphin's material database.

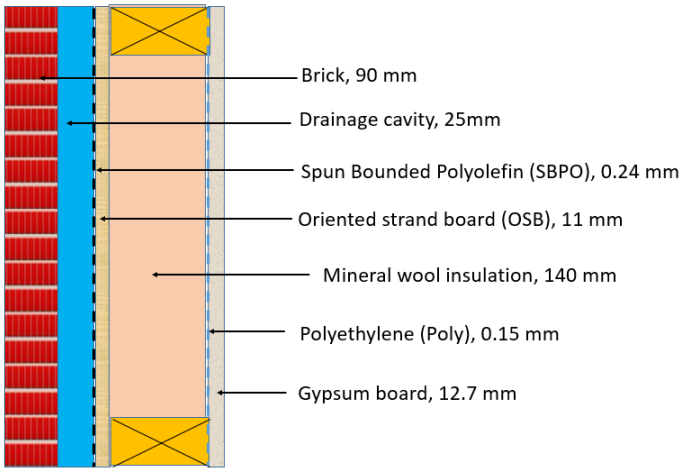


Figure 5 A typical wood-frame wall assembly

Table 1 Basic material properties of the wall components

	RHO[1]	THETA_POR[2]	THATA_EFF[3]	THETA_CAP[4]	μ_dry[5]	A-value[6]	CE[7]	λ[8]
Red matt clay brick	1935	0.265	0.217	0.162	129	0.0268	800	0.5
Air space	1.2	0.99	-	-	1	-	1006	0.025
SBPO[9]	464	0.012	0.012	0.01	305	0.00031	1250	0.248
OSB	650	0.4	0.38	0.27	753	0.0022	1880	0.094
Mineral wool	37	0.92	0.9	0.9	1.09	-	840	0.032
Polyethylene	1256	-	-	-	1·106	-	2100	0.16
Gypsum board	700	0.65	0.42	0.4	138	0.0019	870	0.16

[1]RHO – Bulk density (kg/m^3)
[2]THETA_POR – Porosity (m^3/m^3)
[3]THETA_EFF – Effective saturation water content (m^3/m^3)
[4]THETA_CAP – Capillary water cotent (m^3/m^3)
[5]μ_dry – Vapour resistance factor at dry state (-)
[6]A-value – Water absorption coefficient (kg/m^2.s$^{0.5}$)
[7]CE – Heat capacity (J/kg.K)
[8]λ – Heat conductivity (W/m.K)
[9]SBPO: Spun-bonded polyolefin

To reduce the number of stochastic material properties, a preliminary sensitivity analysis was performed to determine the most influential ones with respect to moisture performance. The details of the sensitivity analysis can be found in (Chetan et al., 2021). It was found that the moisture storage function (scaled by effective saturation water content) and the liquid diffusivity (derived from effective saturation water content and water absorption coefficient) of brick are the most important material properties that influence the mould growth performance on OSB when considering only rain leakage and that the moisture source was deposited on the outer layer of the sheathing membrane. Therefore in this report, the effective saturation water content and the water absorption coefficient of the brick were considered as stochastic material properties.

3.1.2 Boundary conditions and climatic realizations

Table 2 shows the boundary conditions that were considered as constant values. The exterior heat and vapour transfer coefficients were set as dependent on wind speed according to EN ISO 6964 (2017), and the dependency functions are given in equation (7) and (8).

Table 2 Boundary conditions

Interior heat transfer coefficient (W/m^2.K)	Interior vapour transfer coefficient (s/m)	Short wave absorptivity (-)	Long wave emissivity (-)
8	$1.52 \cdot 10^{-8}$	0.6	0.9

$$h_{ce} = 4 + 4 \cdot v \qquad (7)$$

$$\beta_{ve} = 2.44 \cdot 10^{-8} + 2.44 \cdot 10^{-8} \cdot v \qquad (8)$$

Where:

h_{ce} – exterior heat transfer coefficient (W/m^2.K)

β_{ve} – exterior vapour transfer coefficient (s/m)

v – wind speed (m/s)

The wind-driven rain deposited on the exterior wall surface was calculated based on the ASHRAE 160 model as given by equation (9) (ASHRAE, 2016) :

$$r_{bv} = F_E \times F_D \times F_L \times U \times \cos\theta \times r_h \qquad (9)$$

Where:

r_{bv} – wind-driven rain deposited on the exterior wall surface

F_E – rain exposure factor, reflects different exposure types, for buildings lower than 10 m; the rain exposure factor can be assumed to have a value of 1.4 for the severe exposure category, 1.0 for the medium exposure category, and 0.7 for the sheltered category

F_D – rain deposition factor, reflects different roof designs; it can be assumed to have a value of 0.35 for a steep-slope roof, 0.5 for a low-slope roof and 1.0 for a wall subject to rain runoff

F_L – empirical constant, 0.2 kg.s/(m^3.mm)

θ – angle between wind direction and normal to the wall

U – hourly average wind speed at 10 m height, m/s

r_h – rainfall intensity, horizontal surface, mm/h

In this report, the rain exposure factor (FE) was considered as 1, and the rain deposition factor (FD) was considered as a stochastic variable. To simulate rain leakage, a moisture source was deposited on the exterior surface of the spun bonded polyolefin (SBPO). The quantity of this moisture source was considered as a stochastic variable as well. The effect of cladding ventilation was simulated using a source/sink approach, which considers the air in the drainage cavity as a heat and moisture source by imposing a cladding ventilation rate (ACH) that was considered as a stochastic variable.

The 15 climatic realizations of the 31-year climate data sets were generated by Gaur et al. (2019). In this report, climate data that were used for the stochastic simulation included the historical climate data and the climate data with the worst future scenario for global warming (3.5ºC global warming) for the city of Ottawa (Hereafter, referred to as historical and future periods). To save computational costs, a 2-year moisture reference year was selected for each climatic realization based on the moisture index, which was calculated year by year based on equation (10) (Cornick et al., 2003):

$$MI = \sqrt{(1 - DI)^2 + WI^2} \qquad (10)$$

Where

MI – moisture index of a specific year

DI – normalized drying index of a specific year

WI – normalized wetting index of a specific year

The drying index represents the degree of moisture saturation of ambient air, whereas the wetting index considers the wind speed and horizontal rain. The absolute values of yearly WI and DI were calculated using equation (11) and (12) (Cornick et al., 2003):

$$WI = \sum_1^{8760} U \cdot R_h / 1000 \qquad (11)$$

Where:

U – wind speed in a specific hour, m/s

R_h – horizontal rain in a specific hour, mm

$$DI = \sum_1^{8760} (w_{sat} - w) \qquad (12)$$

Where:

w_{sat} – humidity ratio at saturation in a specific hour

w – humidity ratio of ambient condition in a specific hour

In order to consider the relative difference amongst different realizations, the yearly values for DI and WI were normalized based on the maximum and minimum values in all the 15 realizations within one climatological period; thereafter, the moisture index was calculated for every single year for each of the 31-year climatic realizations. The selected 2-year moisture reference year includes one 50 percentile and one 90 percentile of the moisture index in each 31-year climatic realization. Finally, there were 15 realizations (R01, R02 … R15) of 2-year climate data sets for each climatological period, and the 15 realizations of 2-year climate data sets in each period were considered as discrete stochastic variables. Figure 6 shows the 2-year averaged moisture index of each climatic realization in historical and future periods for Ottawa. It can be noted that the MIs were calculated based on DI and WI, which were values normalized within each climatological period. Therefore, the MIs reflect the relative difference amongst different climatic realizations, but, they do not reflect the difference between historical and future periods.

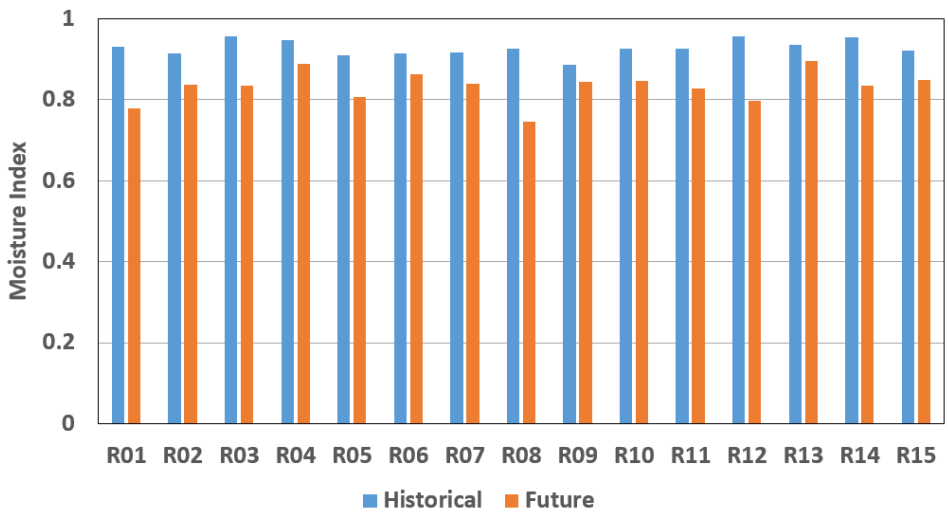


Figure 6 2-year's averaged MIs of each of the 15 climatic realizations

3.2 Stochastic variables and Sobol sequence based sampling

The stochastic variables considered in this report are summarized in Table 3. The details of the quantification of stochastic variables can be found in Wang et al. (2021).

Table 3 Stochastic variables.

Variables	Distribution	Range	Intervals
Climatic realisation	Discrete	R01–R15	-
Orientation	Uniform	0–360	16 Orientations, interval of N 348.75 ~ 11.25, NNE 11.25 ~ 33.75 and so on…
Rain deposition factor	Uniform	0.35–1;	Low 0.35 ~ 0.56; Middle 0.56 ~ 0.78; High 0.78 ~ 1.
Rain leakage moisture source (% of wind-driven rain)	Normal*	0–2.0	Low 0.3 (0.35); Middle 0.56 (0.35); High 0.8 (0.35)
Cladding ventilation rate (ACH)	Normal	1–20	Low 3 (0.67); Middle 5.5 (1.4);

			High 10.5 (3.5)
Water absorption coefficient (A-value)of brick (kg/m2·s0.5)	Normal	0.0161–0.0389	0.0268 (0.005)
Effective saturation water content of brick (m3/m3)	Normal	0.108–0.325	0.217 (0.043)

* For the variables having normal distribution, the values in the parenthesis are standard deviations. The distributions of rain leakage moisture source are truncated above zero; the distributions of cladding ventilation rate are truncated above 1.

To implement the Sobol sequence based sampling, 10 sets of 6-dimensional randomized Sobol sequence were generated, the randomization is based on the scramble approach (L'Ecuyer and Lernieux, 2002) as indicated in Section 2.2.1. The 10 sets of randomized Sobol sequence were generated because the number of sets of randomized Sobol sequence (r in equation (6)) is usually selected as a small constant, when n (the sample size for each Sobol sequence) increases the standard error can be decreased and at an optimized rate (Hou et al., 2019). For continuous stochastic variables, these are mapped into the space [0, 1] through their cumulative distribution, and the values at the percentiles which correspond to the Sobol points are then selected; the implementation of the sampling was illustrated in Section 2.2.2. For discrete variables, i.e. the climatic realizations, the space of [0, 1] was divided into 15 equal intervals, i.e. [0, 1/15], [1/15, 2/15]…… [14/15, 15/15]. When a Sobol point falls into the first interval, then the first realization (R01) is selected; similarly, when a point falls into the second interval, R02 is selected, and so on.

For the rain leakage moisture source and cladding ventilation rate, there were three levels selected, with each level having a specific cumulative distribution. To guarantee that the cumulative distribution at each level is unchanged and the sample sizes at different intervals are equal, the space of [0, 1] was divided into three intervals: [0, 1/3], [1/3, 2/3] and [2/3, 1]. When the Sobol point falls into the first interval, then the low level will be selected, and the stochastic values at that level will be selected by mapping the percentiles in the cumulative distribution to the interval of [0, 1/3]. Similarly, the values in the middle level and high level were also selected in the same way. Figure 7 provides an example of the sample space division based on different categories of rain deposition factor.

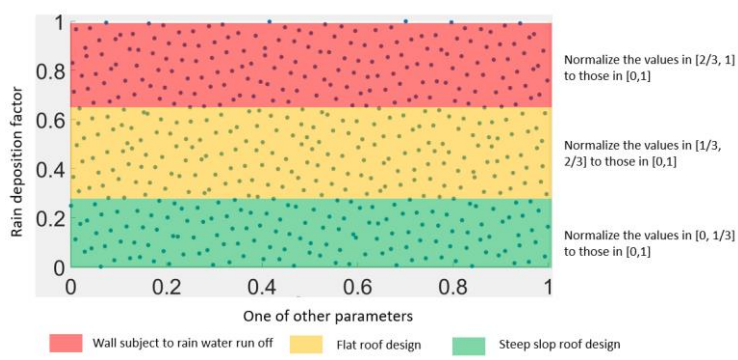


Figure 7 Sample space division based on rain deposition factor

For the rain deposition factor, this was assumed to be uniformly distributed from 0.35 to 1; therefore, it was directly mapped to Sobol points through a uniform cumulative distribution with the sampled values at each category of rain deposition factor also being uniformly distributed. Similarly, the orientation was also assumed as a uniform distribution, and divided into 16 equal intervals, each interval is 22.5º. For example, 11.25º to 33.75º represents NNE, 33.75º to 56.25º represents NE and so on. For the North orientation, this includes two sub-intervals with each having 11.25º: 348.75º to 360º and 0º to 11.25º.

Although there were 7 stochastic variables, the A-value and effective saturation water content of the brick were positively correlated with a correlation coefficient of 0.6. The random number generation of A-values and effective saturation water content of the brick followed that given in Section 2.1, which is the method of generating

correlated stochastic variables; for Sobol based random number sampling, using two columns of Sobol points to sample these two parameters would disrupt their correlation. Therefore, the water penetration coefficient, which is defined as the ratio of A-value to saturation water content (Kunzel, 1995), was used to be sampled by Sobol points. In this way, the two stochastic brick properties can be sampled by one column of Sobol points, and their correlation can be retained.

3.3 Error estimation

The mould growth index was calculated based on the use of the VTT model (Ojanen et al., 2010) with a decline factor of 0.3. The relative humidity and temperature of the exterior surface of OSB were extracted for calculating the mould growth index. The averaged values for mould growth indices over the two years of the simulation were calculated for error estimation and mould growth risk assessment. In this report, the two year average mould growth index of 1 was set as a threshold which should not be exceeded. This differs from that given in ASHRAE 160 (2016) in which it is suggested that to minimize the mould growth risk, the hourly value of mould growth index should not exceed a value of 3. What should be noted is that the average value over two years simulation can only reflect the mould growth risk in such a period, however, it cannot capture the length of the periods when the mould growth index is higher than 3. In the future work, it might be useful to adopt a performance indicator which is able to reflect probability distribution of the mould growth index at different levels over a simulation period. The values for standard error of the mean and standard deviation were calculated based on equations (5) and (6), respectively (Hou et al., 2019).

The error estimation was performed for the entire whole sample space and the sub-spaces for different climatic realizations and orientations at different sample sizes. The sample size of the entire sample space is the number of runs required for stochastic simulations, for which all the stochastic varaibles are considered to vary over their entire range. The sample size of a sub-space is the number of runs required for stochastic simulations in a specific category of a specific environmental or design variable, i.e. the climate realization, the wall orientation or the rain deposition factor. In this report, the error estimation for the entire sample space was performed at sizes of 2^3x10, 2^4x10 up to 2^8x10. For the sub-spaces, the error estimation started from a sample size of 2^5x10 and ended up with 2^8x10, since the sample size of each sub-space is approximately the total sample size divided by the number of intervals. The maximum sample size was determined by considering the balance between the computational cost and the accuracy of the results. As stated in Section 2.4, for the problem of mould index prediction in this report, the acceptable error was limited to ± 0.1.

According to Hou et al.(2019), the standard errors calculated from equations (5) and (6) are roughly in the same order of magnitude as the absolute error that was obtained by comparing the results at different sample sizes (2^3x10, 2^4x10, …, to 2^7x10) with a very large reference sample size - 40960. The results of standard error will be discussed in section 3.4. Then the mould growth risk for different climatic realizations and orientations were assessed based on the results from the highest sample size. For the case study in this report, the entire sample space, the sample size of 2^8x10 in the entire sample space gave a standard error in the magnitude of 10^{-3}, whilst, for each sub-space, the sample size of 2^8x10 gave a standard error in the magnitude of 10^{-2}. Details in respect to the standard error decreasing with the increase of sample size will be presented in Figure 8 and Figure 9.

3.4 Results and analysis

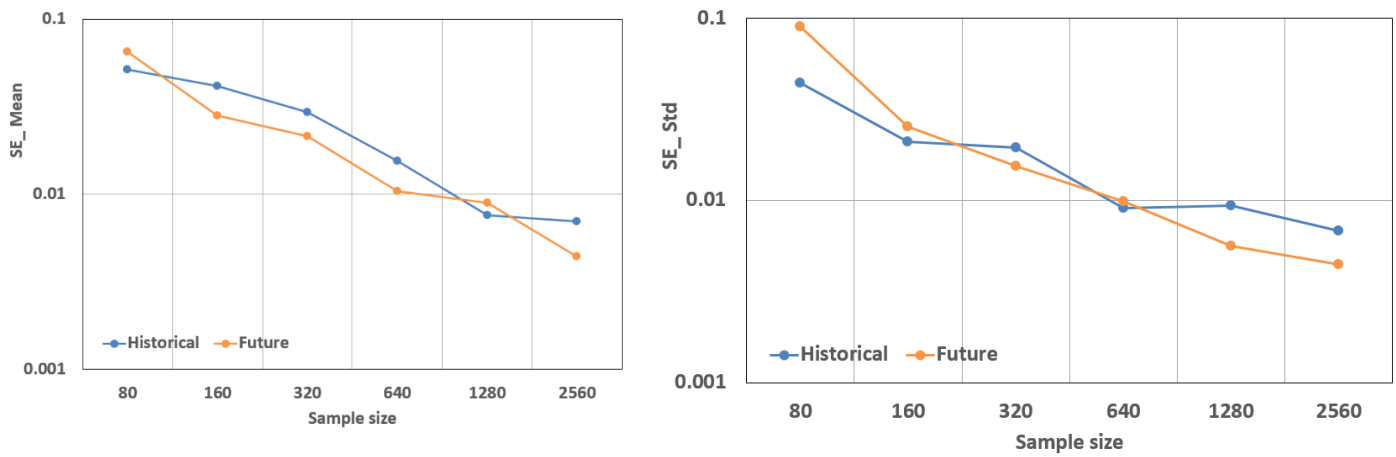
This section presents two aspects of the stochastic results, the convergence rate of standard error- the decreasing rate of the standard error with the increase of the sample size, and the mould index at different climatic realizations, wall orientations and risk mitigation strategies. The convergence rate analysis was performed for different climatic realizations and wall orientations to demonstrate the methodology of error estimation. Then, the mould growth risk assessment was implemented for different climatic realizations and wall orientations based on the results from the maximum sample size. The mould index for different risk mitigation

strategies was evaluated based on the worst orientation, which has the highest mould index level based on the stochastic results from the maximum sample size.

3.4.1 Convergence rate of standard error

3.4.1.1 The whole sample space

Figure 8 shows the standard error of the mean value and standard deviation of the 2-year averaged mould growth index at different sample sizes (number of runs) for the entire sample space, which means all the climatic realisations and orientations, as well as other stochastic variables, were taken into account. It can be seen that the standard error generally decreased with an increase in sample size, although the decreasing rates are different at different sample size intervals, i.e., for the mean value of the historical period, the decreasing rate is greater from 320 to 1280 than from 80 to 320. When the sample size is increased to 1280-runs, the standard error can be reduced to a magnitude of 10^{-3} for both historical and future periods. As shown in Table 4, the future period has a higher mould growth index than the historical period, but the standard deviation in the future is very similar to that in the historical period. The standard error in the future period is slightly lower than that in the historical period. The mould growth index listed in Table 4 reflects the overall mould growth risk in the city of Ottawa, considering all the climatic realisations, orientations and other stochastic variables.



(a) SE of Mean values (b) SE of Standard deviations

Figure 8 Standard error (SE) of mean and standard deviation (std) values at different sample size

Table 4 Mean value and standard deviation of 2-year averaged mould index at a 2560-run.

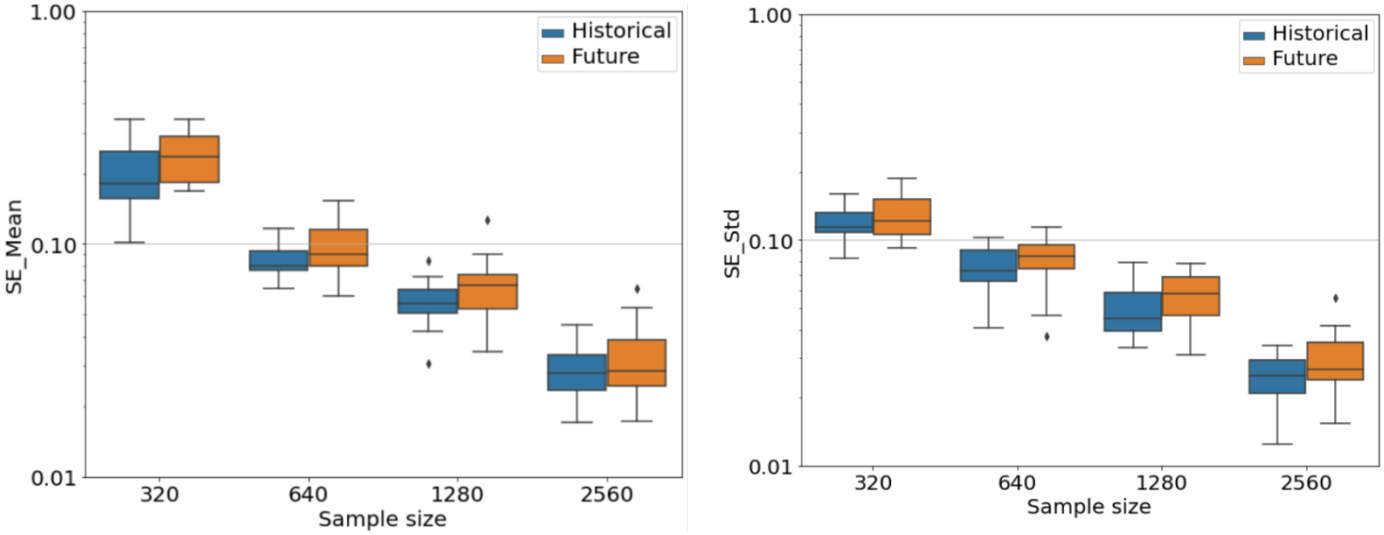
	Historical Period	Future Period
Mean value (SE)	1.06 (0.007)	1.44 (0.004)
Standard deviation (SE)	0.91 (0.007)	1.08 (0.004)

SE: standard error

3.4.1.2 Different climatic realizations

Figure 9 shows the standard errors at different sample sizes for different climatic realisations of historical and future periods. Each box represents the standard errors of 15 climatic realisations in a climatological period. In general, the standard errors for different climatic realisations are higher than those for the entire sample space in the same sample size of the entire space, since the entire sample space was divided into different sub-spaces based on climatic realizations and the sample size for each climatic realisation is roughly equal to the total sample size divided by 15. It can be seen that there is a significant variation of the standard error for different climatic

realisations, which implies that different climatic realizations require different number of runs to achieve the same level of accuracy in the results. The standard errors for the future period are generally higher than those for the historical period, because in the future period, there are more uncertainties in climatic conditions such as more heat waves and extreme rain events, and such events will increase the uncertainties of the wall performance. Therefore, the simulations for the future period require more simulation runs to obtain the same level of accuracy as compared to the simulations for the historical period. The standard errors of mean values and standard deviation values can be reduced to below 0.1 after 1280 runs, except for the mean values of the future period, which require 2560 runs to ensure all the standard errors fall to below 0.1.



(a)SE of mean value

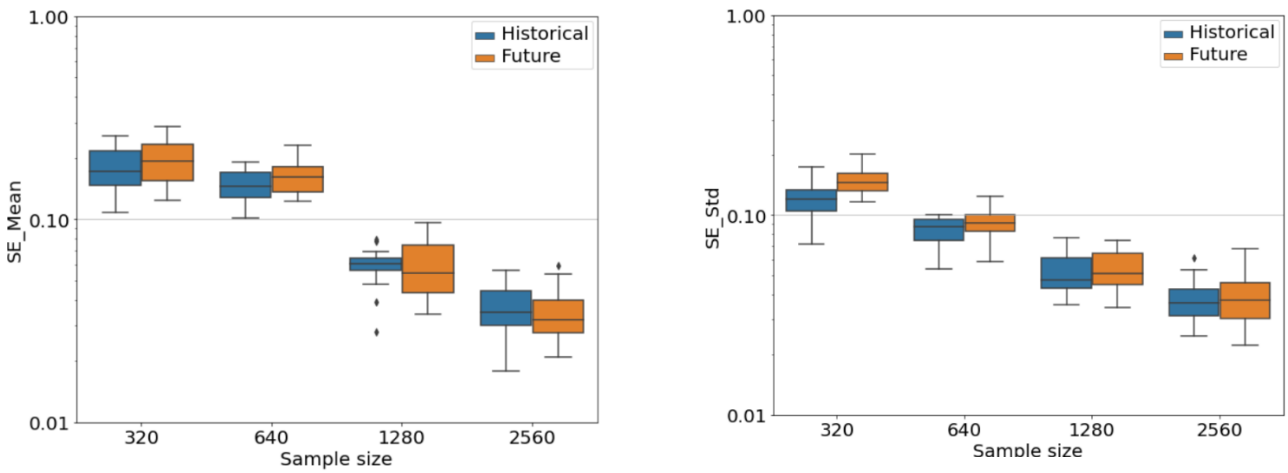
(b) SE of standard deviation

Figure 9 Convergence rate of standard error (SE) for different climatic realizations

Note: there are 15 climatic realisations, therefore, the sample size for each realisation is roughly equal to the total sample size divided by 15. The marks of "♦" are outliers.

3.4.1.3 Different Orientations

Figure 10 shows the standard errors of the mean value and standard deviation of different orientations for the historical and future periods: each box represents the standard errors for 16 orientations in each climatological period. It can be seen that the decreasing rates of the standard errors for different orientations are lower than those for different climatic realisations, particularly from 320-run to 640-run. Future period generally has higher standard errors than historical period. The variation of standard error amongst different orientations is significant. The standard errors of all the orientations can be reduced to below 0.1 after 1280 runs, which means for each orientation 80 runs can achieve a standard error lower than 0.1.



(a) SE of mean value

(b) SE of standard deviation

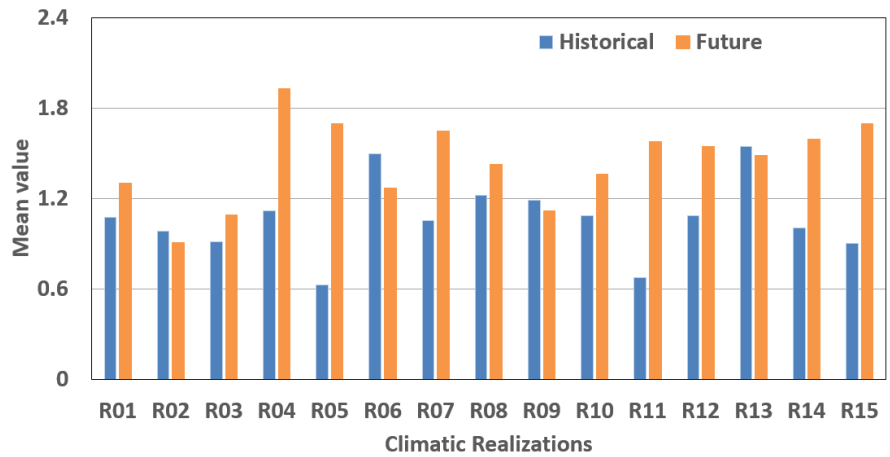
Figure 10 Convergence rate of standard error (SE) for different orientations.

Note: there are 16 orientations, therefore, the sample size for each orientation is roughly equal to the total sample size divided by 16. The marks of "⧫" are outliers.

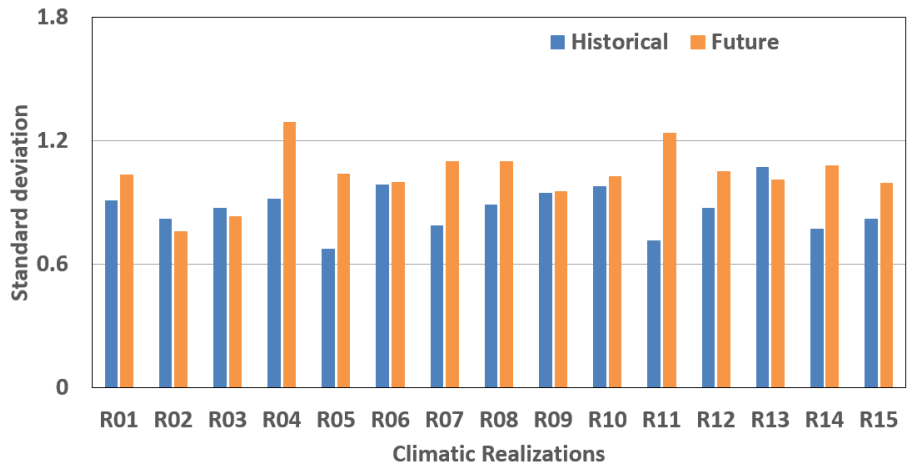
3.4.2 Mould growth risk assessment for different climatic realizations and wall orientations

3.4.2.1 Different climatic realizations

Figure 11 shows the mean value and standard deviation of the 2-year averaged mould growth index for historical and future periods using 2560-runs for which the standard error for each climatic realisation is below 0.1. It can be seen that most of the climatic realisations have a higher mould growth index in the future than in the historical period. The future period also has higher variability in the mould growth index than the historical period given the higher standard deviation, which might be caused by higher uncertainties of the future climate data. The higher uncertainties of future climate data, such as more extreme weather events, have increased the uncertainties of the mould growth index given the moisture loads imposed on the wall assembly are subjected to a higher variability in the future. As shown in Figure 12 a,b, there is no clear relationship between mould growth index and moisture index based on the stochastic simulation results when all the orientations were taken into account. However, the significance of the relationship increased when the analysis only focused on the orientation that receives the highest amount of WDR (The sample size was locally expanded to 1280-runs for the worst orientation to guarantee the values standard error were below 0.1).

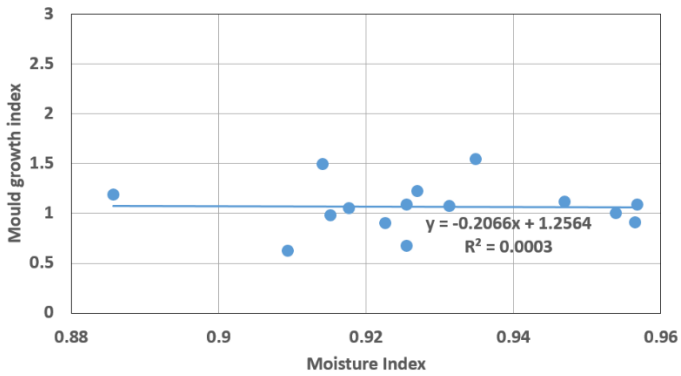


(a) Mean value

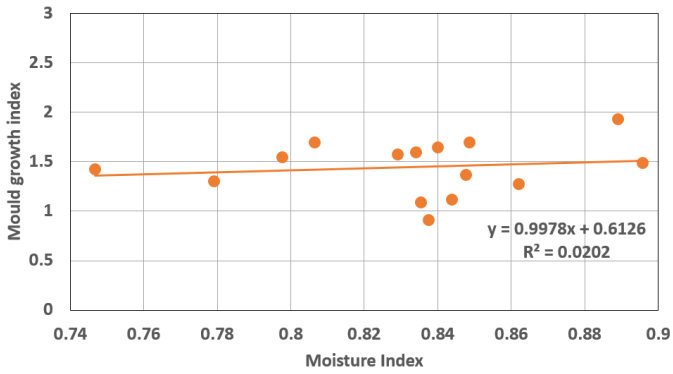


(b) Standard deviation

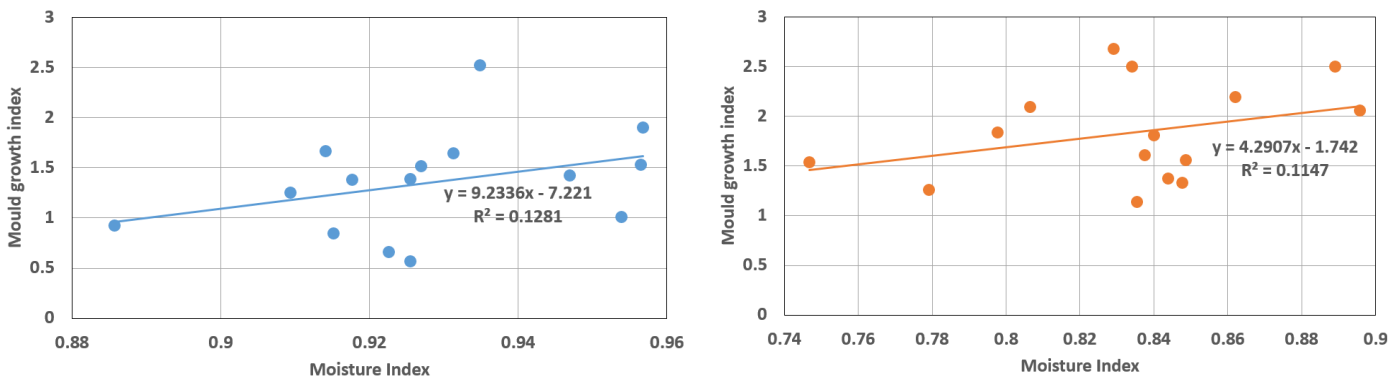
Figure 11 Mould growth index (2-year's average) of 15 climatic realizations using 2560 runs



(a) Historical period



(b) Future period

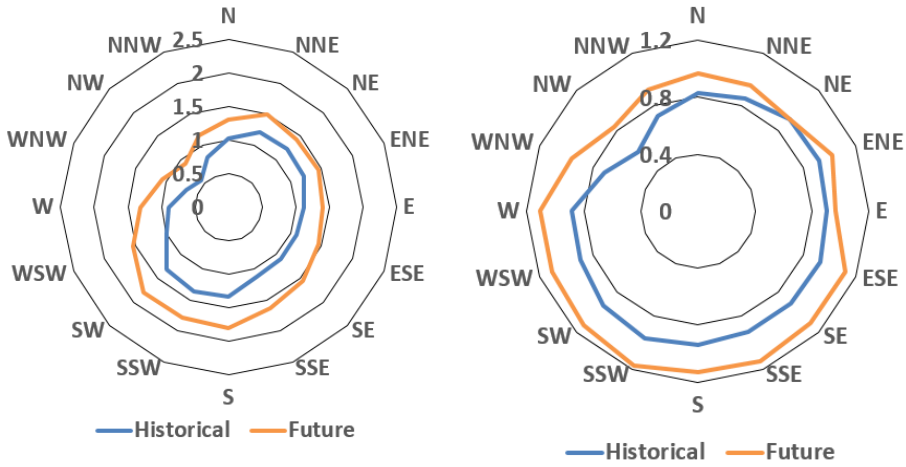


(c) Historical period_ worst orientation (d) Future period_ worst orientation

Figure 12 Mould growth index (mean value of 2-year's average) as a function of moisture index

3.4.2.2 Different wall orientations

Figure 13 shows the mean value and standard deviation of the 2-year averaged mould growth index for different orientations using 2560 runs, which has a standard error of around 0.05 for all orientations. It can be seen that the mean value and standard deviation of the mould growth index for the future period are higher than those for the historical period for all orientations. The orientation that had the highest mean value and standard deviation of mould growth index was SSW. The SSW orientation was selected as the orientation for further stochastic simulations and the one providing the most severe conditions to which to subject wall assemblies when investigating the mould growth risk under different mitigation strategies.



(a) Mean value (b) Standard deviation

Figure 13 Mould growth index (2-year's average) for different orientations at 2560-run.

3.4.3 Mould growth risk assessment for different risk mitigation strategies

To investigate different mould growth risk mitigation strategies, an additional 640 stochastic simulations were performed for the worst orientation, as selected in Section 3.4.3. Figure 13 shows the stochastic 2-year averaged mould growth index for different mould growth risk variables and variable levels, i.e., the rain deposition factor, rain leakage moisture source and cladding ventilation rate as described in Table 9. Each box represents the stochastic 2-year averaged mould growth index of one level of a specific mould risk variable with all the other stochastic variables varying in their full range of values. For example, the green box in the case of rain deposition

factor (FD) represents a low level of rain deposition factor (0.35 ~ 0.56) with all the other stochastic variables (climatic realisations, rain leakage moisture source, cladding ventilation rate, and brick properties) varying in their full range of values. As such, if the mould growth index can nonetheless be reduced by controlling one specific variable in consideration of all the uncertainties, the influence of this variable can be considered as a robust control strategy. The 640-run stochastic simulation based on RQMC gave the value for the standard error of the mean and standard deviation for the mould index at each level for each risk variable of ca. 0.05.

3.4.3.1 The influence of rain deposition factor and cladding ventilation rate

In Figure 14, it can be seen that the reduction of the rain deposition factor and increase of ventilation rate can significantly reduce the mould growth index, and the uncertainty at a low level of rain deposition factor is lower than that at the high level of cladding ventilation rate, particularly in the future period which has higher moisture loads and uncertainties of extreme weather events. This indicates that, under the climatic conditions of Ottawa, controlling the rain deposition factor could be a more robust mitigation strategy than increasing the cladding ventilation rate. The same stochastic simulations were also implemented for Halifax, a coastal city in Canada. It was found that ACH plays a more important role in controlling mould growth than the rain deposition factor. Therefore, the most robust design or risk mitigation strategy varies depending on the climatic region.

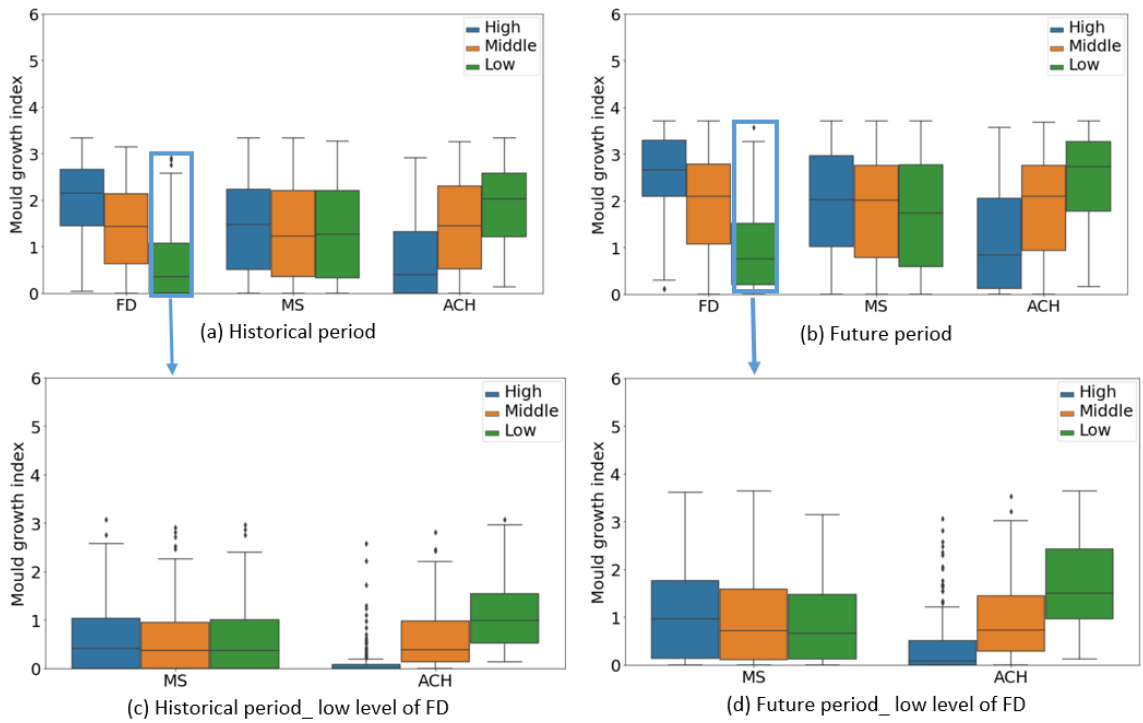


Figure 14 Boxplots of 2-year's averaged mould growth index at different levels of mould growth risk control variables-MS on exterior WRB.

FD: Rain deposition factor; MS: Rain leakage moisture source; ACH: Cladding ventilation rate. For ACH, the high level is the 4 opening ventilated design, the middle level is the 2 opening ventilated design and the low level is the vented cavity which has no opening on the top. The marks of "⧫" are outliers.

According to Kubilay et al. (2017), for mid-rise building, even a small building façade detail, such as window sills with a size of 0.1 m, can reduce the catch ratio by up to 37% as compared to the building without any protection, and can achieve a low level of rain deposition factor (Wang et al. 2021). As to low-rise building, the 0.1 m roof-overhang could produce the same protection effect as the 0.1 m window sills of mid-rise building. In the historical period, a low-level of rain deposition factor can reduce the 75 percentile value for the 2-year average mould

growth index to below 1, and 1.5 for the future period. On the other hand, the highest level of the cladding ventilation rate (ventilated cavity with 4 openings as calculated in Wang et al. (2021)) can, for the historical period, reduce the 75 percentile value for the 2-year average mould growth index to below 1.2 and for the future period to a value below 2. Since the 0.1-m overhang seems to be the most robust strategy in controlling mould growth in Ottawa, further simulations were performed for a 0.1-m overhang with different levels of moisture source and ACH, for which the results are presented in Figure 14 c,d. It can be seen that for a building with a 0.1-m roof overhang and a wall with a ventilation design having 4 openings, the 2-year average mould growth index can be controlled within a value of 1 for 90% of the stochastic cases in both historical and future climate periods, although the mould growth index is higher in the future period than in historical period. According to the filed measurement by Chiu (2016), for mid-rise buildings, the roof-overhang can effectively protect the façade area right below the roof, however, the effectiveness of overhang protection was significantly reduced for the area far away from the roof.

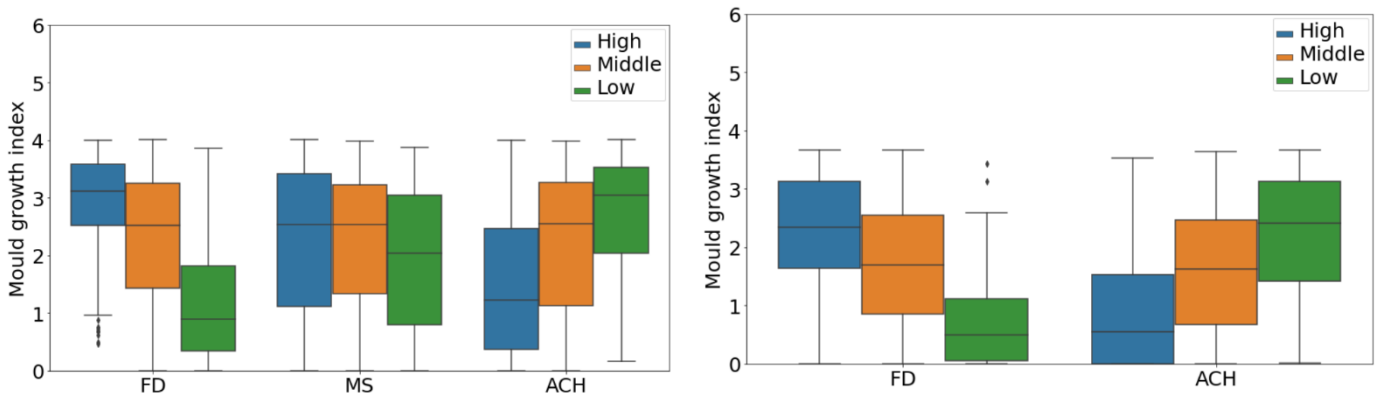
3.4.3.2 The influence of rain leakage moisture source

There is no significant difference between different levels of moisture source from rain leakage, although the mould growth index for the low level of MS is slightly lower than that for the middle and high levels of MS (Figure 14). This maybe due to the protection provided by the water-resistive barrier. However, the uncertainty within each level of MS is very high (Figure 14a,b). The uncertainty is caused by other stochastic variables, i.e., rain deposition factor, ACH, climatic realisations and brick properties. The uncertainties for each level of MS were reduced within the low level of rain deposition factor (Figure 14c, d).

The rain leakage modelling approach was based on ASHRAE 160 (2016), which assumes the moisture source on the exterior surface of the sheathing membrane. However, the rainwater may penetrate through the sheathing membrane and reach the OSB sheathing if there is a defect in the sheathing membrane. To consider this situation, an additional set of simulations was performed with the moisture source assigned to the exterior layer of OSB, that represents a higher mould growth risk scenario than the moisture source assigned to the sheathing membrane for the future period (Figure 15a). As opposed to the high-risk scenario, for comparison, another scenario without any rain leakage was simulated based on the future period (Figure 15b).

When the moisture source was deposited on the exterior layer of the OSB (Figure 15a), the difference between different levels of moisture source is more significant than in the scenario with the moisture source deposited on the exterior surface of the sheathing membrane (Figure 14b). The overall mould growth indices increased for the low level of rain deposition factor and high level of cladding ventilation rate. However, the increase of mould growth index in the low level of rain deposition factor is lower than that in the high level of cladding ventilation rate. For example, the 75 percentile mould growth index at a low level of FD increased from 1.5 to 1.8, and this value at the high level of ACH increased from 2.0 to 2.7. (e.g., compare results given in Figure 14b to those in Figure 15a).

For the scenario without moisture source, the overall mould growth indices at different levels of FD and ACH were reduced (Figure 15b). The 75 percentile mould growth index at a low level of FD decreased from 1.5 to 1, and that for a high level of ACH decreased from 2 to 1.5 (Figure 14b compare to Figure 15b). Therefore, it can be said that improvements in water tightness can overally reduce mould growth risk.



(a) MS on the exterior layer of OSB

(b) No MS

Figure 15 Boxplots of 2-year's averaged mould growth index at different levels of mould growth risk control variables-comparison between different locations of MS.

Note: The marks of "⧫" are outliers.

4 Conclusions

The use of a stochastic approach has been applied for assessing building performance from simulations now for a few decades, and it continues to show its usefulness in uncertainty analysis and risk assessment given the increase in computational capacity, particularly in the current setting where climate change brings more uncertainties to results from hygrothermal simulation. In this report, a generalized framework of stochastic simulation of hygrothermal performance of wood-frame building envelopes was provided, and the implementation of this framework was illustrated with the aid of Python scripts. In addition, a case study was provided to demonstrate the application of the stochastic simulation framework. In summary, the primary functionalities of this framework include:

- It is able to consider the uncertainties of different types of input parameters, including material properties, boundary conditions, heat/moisture sources, and climate data;
- The correlated parameters can be taken into account when generating the random numbers from probabilty distributions;
- The Sobol sequence based sampling provides a high sampling efficiency, and the randomized Sobol sequences allow the users to estimate the standard error of the stochastic results at different sample sizes;
- The sampling process allows the users to assess the hygrothermal performance of building envelopes for different climate scenarios and design scenarios, given that the sample space can be divided into different sub-spaces, which represent different categories of climate or design scenarios.

The functionalities mentioned above allow practitioners to evaluate moisture damage risks at different climatic conditions for different design or risk mitigation strategies. In addition, this stochastic simulation framework, or indeed, some of the functionalities in the framework can be adapted to overheating risk assessment.

However, there are still limitations of this stochastic simulation framework. These limitations might need to be addressed in future work to obtain a more reliable assessment of the climate resilience of building envelopes:

- This stochastic simulation framework did not consider the model uncertainty of the material properties. For example, there are different ways to derive the liquid diffusivity curves, it is still not clear whether the uncertainty caused by the different derived equations would induce a conclusive difference in the moisture damage risk assessment or not. In addition, the temperature dependency of the hygorthermal properties was

not taken into consideration, because of the limitation of the software and the lack of information regarding the temperature dependent moisture storage functions, moisture transport properties and thermal conductivities.

- The aging in material properties i.e. the degradation of the properties of brick, membrane and OSB caused by exposure to prolonged environmental loads, was not considered in the stochastic simulation framewok. The degradation in material properties is greatly related to the severity levels in climate conditions. However, in the current stochastic simulation framework, the material properties were considered as constant values over the simulation period, and thus one cannot evaluate the impact of the climate conditions on the aging of the material properties.
- In the current stochastic simulation framework, the cladding ventilation rate (ACH) was considered as a constant variable during the simulation time period. However, the ACH is a stochastic time series which is highly correlated with wind and buoyancy pressure.
- Although the absolute values for rain leakage moisture source (kg/m^3.s) are time series parameters used for simulation, their randomness reflected the influence of wind pressure and rain intensity acting on the exterior of the building envelope. However, the rain leakage rate (% or wind-driven rain) was considered as a constant value throughout the simulation period for each stochastic model. The joint probability or co-occurrence frequency of high wind-driven rain/ rain leakage rate and low ACH, which might be related to the moisture damage risks of the wall assemblies, cannot be revealed using the current framework.

References

ASHRAE. (2016). ASHRAE Standard 160-2016. Criteria for Moisture-Control Design Analysis in Buildings. Atlanta: ASHRAE.

Chetan, A., Wang, L., Defo, M., Ge, H., Junginger, M., & Lacasse, M.A. (2021). Sensitivity analysis of hygrothermal performance of wood framed wall assembly under different climatic conditions: The impact of cladding properties. The 8th International Building Physics Conference. Copenhagen, Demark.

Chiu, V. (2016). The effect of overhang on wind-driven rain wetting for a mid-rise building. Master's Thesis. Concordia University, Montreal, Canada.

Cornick, S., Djebbar, R., & Dalgliesh, W.A. (2003). Selecting moisture reference years using a moisture index approach. Building and Environment 38, 1367–1379.

Defo, M., Wang, L., & Lacasse, M.A. (2021). Results from Hygrothermal Simulations and on Durability and Resilience of Wall Assemblies to Climate Change. Ottawa, ON, Canada: National Research Council of Canada.

EN ISO 6946. (2017). Building Components and Building Elements—Thermal Resistance and Thermal Transmittance—Calculation Methods. Brussels, Belgium: CEN.

Gaur, A., Lacasse, M., & Armstrong, M. (2019). Climate data to undertake hygrothermal and whole building simulations under projected climate change influences for 11 Canadian cities. *Data, 4*(2), 72. doi:10.3390/data4020072

Hou, T., Nuyens D., Roels, S., & Janssen, H. (2017). Quasi-Monte-Carlo-based probabilistic assessment of wall heat loss. The 11th Nordic Symposium on Building Physics, Trondheim, Norway.

Hou, T., Nuyens, D., Roels, S., & Janssen, H. (2019). Quasi-Monte Carlo based uncertainty analysis: sampling efficiency and error estimation in engineering applications. *Reliability Engineering and System Safety*, 191, 106549.

IPCC, 2021: Climate Change 2021: The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change [Masson-Delmotte, V., P. Zhai, A. Pirani, S.L. Connors, C. Pean, S. Berger, N. Caud, Y. Chen, L. Goldfarb, M.I.Gomis, M. Huang, K. Leitzell, E. Lonnoy, J.B.R. Matthews, T.K. Maycock, T. Waterfield, O. Yelekci, R. Yu, and B. Zhou (eds.)]. Cambridge University Press. In Press

Kumaran, M. K., Lackey, J., Normandin, N., van Reenen, D., & Tariku, F. (2002). Summary report from task 3 of MEWS project at the Institute for Research in Construction - Hygrothermal properties of several building materials. Report IRC-RR-110. Ottawa: NRC.

Kubilay, A., Carmeliet, J., & Derome, D. (2017). Computational fluid dynamics simulations of wind-driven rain on a mid-rise residential building with various types of façade details. *Journal of Building Performance Simulation*. 10, 125–143.

Kumaran, M.K., Lackey, J.C., Normandin, N., & van Reenen, D. (2006). Vapour permeances, air permeances and water absorption coefficients of building membranes. *Journal of Testing and Evaluation*, 34 (3), 241–245.

Kunzel, H.M. (1995) Simultaneous heat and moisture transport in building components. Ph.D. Thesis, Fraunhofer Institute of Building Physics, Fraunhofer, German.

Lacasse, M. A., Ge, H., Hegel, M., Jutras, R., Laouadi, A., Sturgeon, G., & Wells, J. (2018). Guideline on Design for Durability of Building Envelopes. Ottawa, ON, Canada: National Research Council of Canada.

L'Ecuyer, P., & Lemieux, C. (2002). Recent advances in randomized Quasi-Monte Carlo methods. In M. Dror, P. L'Ecuyer, F. Szidarovszky (Eds), *Modelling Uncertainty- An Examination of Stochastic Theory, Methods, and Applications* (pp. 419-474). SpringerLink.

Maref, W., Lacasse, M.A., & Booth, D.G. (2003). Assessing the hygrothermal response of wood sheathing and combined membrane-sheathing assemblies to steady-state environmental conditions. *The 2th International Building Physics Conference*. Leuven, Belgium.

Ojanen, T., Viitanen, H., Peuhkuri, R., Lâhdesmäki, K., Vinha, J., & Salminen, K. (2010). Mold growth modeling of building structures using sensitivity classes of materials. *Pro. Building XI.* Florida.

PyTorch. (2021). Retrieved from http://pytoch.org

Saltelli, A., Annoni, P., Azzini, I., Campolongo, F., Ratto, M., & Tarantola, S. (2010). Variance based sensitivity analysis of model output. Design and Estimator for the total sensitivity index. Computer Physics Communications, 181(2), 259-270.

Simpson, Y. (2010). Field evaluation of ventilation wetting and drying of rainscreen walls in coastal British Columbia. Master's Thesis. Concordia University. Montreal, Canada.

Wang, L., Defo, M., Xiao, Z., Ge, H., & Lacasse, M.A. (2021). Stochastic simulation of mould growth performance of wood-frame building envelopes under climate change: risk assessment and error estimation. *Buildings*, 11, 333.

Wikipedia. (2021). Retrieved from https://en.wikipedia.org/wiki/Sobol_sequence

Zhang, X., Flato, G., Kirchmeirer-Yong, M., Vincent, L., Wan, H., Wang, X., Rong, R., Fyfe, J., Li, G., & Kharin, V.V. (2019). Changes in temperature and precipitation across Canada. In E. Bush & D.S.Lemmen (Eds), *Canada's Changing Climate Report* (pp. 112-193). Ottawa, ON, Canada : Government of Canada

Zhao, J., Plagge, R., Ramos, N.M.M., Simoes, M.L., & Grunewald, J. (2015). Concept for development of stochastic databases for building performance simulation- a material database pilot project. *Building and Environment*, 84, 189-203

Appendix A. Scripts for generating random numbers from different probability distributions

1. Functions for generating random numbers from different probability distributions

```python
import numpy as np
import scipy.stats as stats

# This function generates random number from  uniform distribution
# The inputs are: lower and upper values of the uniform distribution, and sample size n
# The output is a numpy array including random numbers from a uniform distribution
def generate_random_number_uniform (lower, upper, n):
    np_random_uniform = np.random.uniform (lower,upper,n)
    return np_random_uniform

# This function generates random number from  normal distribution
# The inputs are: mean value and standard deviation of the normal distribution, and sample size n
# The output is a numpy array including random numbers from a normal distribution
def generate_random_number_normal (mean, std, n):
    np_random_normal = np.random.normal (mean,std,n)
    return np_random_normal


# This function generates random number from  normal distribution, wtih truncated bounds
# The inputs are: mean value and standard deviation of the normal distribution,
# as well as the lower and upper bounds, and sample size n
# The output is a numpy array including random numbers from a truncated normal distribution
def generate_random_number_normal_truncated (mean, std, lower_bound, upper_bound, n):
    normal_truncated = stats.truncnorm((lower_bound - mean)/ std, (upper_bound - mean)/ std, loc = mean, scale = std)
    np_random_normal_truncated_samples =  normal_truncated.rvs(n)
    return np_random_normal_truncated_samples
```

```python

# This function generates random number from  two correlated normal distributions
# The inputs are: mean value and standard deviation of two normal distributions,
# as well as the correlation coefficient corr, and sample size n
# The output is a  numpy array including random numbers generated from two correlated normal distributions
def generate_random_number_normal_correlated (mean1, std1, mean2, std2, corr, n):
    np_mean =  np.array([mean1,mean2])
    np_std =   np.array([[std1,0],[0,std2]])

    np_corr = np.array([[1,corr],[corr,1]])

    np_cov = np.matmul(np_std,np_corr )
    np_cov = np.matmul(np_cov,np_std )

    np_normal_correlated = np.random.multivariate_normal (np_mean,np_cov,n)

    return np_normal_correlated

# This function generates evenly distributed intervals between 0 and 1
# The input is n, the number of levels of discrete variable, for example, 15 climate realization will have n = 15
def generate_random_number_discrete (n):
    np_random = np.arange(0,1,1/n)
    return np_random
```

2. Example for generating random numbers from stochastic variables

```python
# This script is the main routine of generating random values from specific probability distributions
from RandomNumber import generate_random_number_uniform
from RandomNumber import generate_random_number_normal_truncated
from RandomNumber import generate_random_number_normal_correlated
from RandomNumber import generate_random_number_discrete
import pandas as pd
import numpy as np
from Plots import make_figure

# Generate discrete variable- climatic realizations
random_Climate_Realizations = generate_random_number_discrete (15)

# Generate continuous uniformly distributed randome variable- Wall Orientation
random_Orientation = generate_random_number_uniform(0,360,10000)

# Generate continuous uniformly distributed randome variable- Rain deposition factor
random_FD = generate_random_number_uniform(0.35,1,10000)

# Generate continuous normaly distributed randome variable- Rain leakage moisture source
# The random variable of rain leakage moisture source was divided into three categories: Low, Middle and High
random_MS_Low = generate_random_number_normal_truncated (0.3,0.35,0,2,10000)
random_MS_Middle = generate_random_number_normal_truncated (0.56,0.35,0,2,10000)
random_MS_High = generate_random_number_normal_truncated (0.8,0.35,0,2,10000)

# Generate continuous normaly distributed randome variable- Cladding ventilation rate ACH
# The random variable of ACH was divided into three categories: Low, Middle and High
random_ACH_Low = generate_random_number_normal_truncated (3,0.67,1,5,10000)
random_ACH_Middle = generate_random_number_normal_truncated (5.5,1.83,1,10,10000)
random_ACH_High = generate_random_number_normal_truncated (10.5,3.5,1,20,10000)

# Generate correlated normaly distributed random variables: effective saturation water content and water absorption coefficient
random_Brick_Properties = generate_random_number_normal_correlated(0.0268,0.0035,216.7,29,0.6,10000)

# Check hisgogram of stochastic variables. Following script shows an example of the histogram plotting of stochastic variables
fig_hist,ax_hist = make_figure ('ACH_Low', 'Probability density', 25)
ax_hist.hist(random_ACH_Low, bins = 20, edgecolor='black', linewidth=1.2, density = True)

# Check correlation between effective saturation water content and A-value
fig_brick,ax_brick = make_figure ('Thetal', 'A-value', 25)
ax_brick.scatter(random_Brick_Properties[:,1],random_Brick_Properties[:,0])

# Save the random numbers to a specific directory
df_random_Climate_Realizations = pd.DataFrame(data = random_Climate_Realizations)
df_random_Climate_Realizations.to_csv\
    ('D:/StochasticModelling/StochasticScripts/stochastic_variables/stochastic_variables_discrete_climate.csv')

list_column_continuous_variables = ['Orientation','FD','MS_Low','MS_Middle','MS_High','ACH_Low','ACH_Middle','ACH_High','A_Brick',
                                    'Theta_Brick']
np_stochastic_variables = np.empty([10000,10])
np_stochastic_variables[:,0] = random_Orientation
np_stochastic_variables[:,1] = random_FD
np_stochastic_variables[:,2] = random_MS_Low
np_stochastic_variables[:,3] = random_MS_Middle
np_stochastic_variables[:,4] = random_MS_High
np_stochastic_variables[:,5] = random_ACH_Low
np_stochastic_variables[:,6] = random_ACH_Middle
np_stochastic_variables[:,7] = random_ACH_High
np_stochastic_variables[:,8] = random_Brick_Properties[:,0]
np_stochastic_variables[:,9] = random_Brick_Properties[:,1]

df_stochastic_variables = pd.DataFrame (data = np_stochastic_variables, columns =  list_column_continuous_variables)
df_stochastic_variables.to_csv ('D:/StochasticModelling/StochasticScripts/stochastic_variables/stochastic_variables_continuous.csv')
```

Appendix B. Scripts for generating randomized Sobol points

```python
# This function generates r groups of randumized sobol points with  dimension of d and maximum sample size of n
# The inputs of the function are:
# r, the number of sets of sobol points;
# d, the dimension of sobol points; n, the sample size of the sobol points
import torch
import pandas as pd

def generate_sobol_points (r,d,n):
    list_sobol_points = list()
    for i in range(r):
        sobol_sequence = torch.quasirandom.SobolEngine(dimension = d, scramble = True, seed = None)
        sobol_points =  sobol_sequence.draw (n)
        np_sobol_points = torch.Tensor.numpy(sobol_points)
        pd_sobol_points = pd.DataFrame (data = np_sobol_points)
        list_sobol_points.append(pd_sobol_points)

    return list_sobol_points
```

Appendix C Scripts for sampling random numbers based on Sobol points

```python
import pandas as pd
import numpy as np

# Load sobol points
directory_sobol_points = 'D:/StochasticModelling/StochasticScripts/sobol_points/sobol_points_'

list_sobol_points = list()
r = 10
sample_size = 32
for i in range (r):
    df_sobol_points = pd.read_csv (directory_sobol_points + str(i)+'.csv')
    df_sobol_points.set_index('Unnamed: 0',inplace = True)
    np_sobol_points = df_sobol_points.to_numpy()
    list_sobol_points.append (np_sobol_points)

# Load stochastic variables
directory_stochastic_variables_discrete = \
    'D:/StochasticModelling/StochasticScripts/stochastic_variables/stochastic_variables_discrete_climate.csv'
directory_stochastic_variables_continuous = \
    'D:/StochasticModelling/StochasticScripts/stochastic_variables/stochastic_variables_continuous.csv'

df_stochastic_variables_discrete = pd.read_csv (directory_stochastic_variables_discrete)
df_stochastic_variables_discrete.set_index('Unnamed: 0',inplace = True)
df_stochastic_variables_continuous = pd.read_csv (directory_stochastic_variables_continuous)
df_stochastic_variables_continuous.set_index('Unnamed: 0',inplace = True)
np_stochastic_variables_continuous=df_stochastic_variables_continuous.to_numpy()

# Define numpy array of sobol stochastic variables - the randum numbers sampled by sobol points
np_sobol_stochastic = np.empty([sample_size,7])
# Define numpy array of the categories of sobol stochastic variables
np_sobol_stochastic_categories = np.empty([sample_size,5], dtype = 'object')
```

```python

# Define a list to store different sets of sobol stochastic variables
list_sobol_stochastic = list()
# Define a list to store different sets of the categories of sobol stochastic variables
list_sobol_stochastic_categories = list()

# Pre-processing of correlated parameters: effective saturation water content and A-value
# Calculate water penetration coefficient: np_B is an array of water panetration coefficient
np_B = np_stochastic_variables_continuous[:,8]/np_stochastic_variables_continuous[:,9]

np_sobol_sampled_B = np.empty([sample_size,1])
np_diff_sobol_original_B = np.empty([10000,sample_size])
np_sampled_A_Theta_index = np.empty([sample_size,1], dtype = 'int16')

list_sobol_sampled_B = list()
list_diff_sobol_original_B = list()
list_sampled_A_Theta_index = list()

# Sample water penetration coefficient based on the last column of sobol point
for i in range(r):
    for j in range(sample_size):
        np_sobol_sampled_B[j] = np.percentile(np_B,100*list_sobol_points[i][j,5])
    list_sobol_sampled_B.append(np_sobol_sampled_B)
    np_sobol_sampled_B = np.empty([sample_size,1])

# Extract the indices of the sampled effective saturation water content and A-value
for i in range (r):
    for j in range(sample_size):
        np_diff_sobol_original_B = abs(list_sobol_sampled_B[i][j] - np_B)
        np_sampled_A_Theta_index[j] = np.argmin(np_diff_sobol_original_B)
    list_sampled_A_Theta_index.append(np_sampled_A_Theta_index)
    np_sampled_A_Theta_index = np.empty([sample_size,1], dtype = 'int16')
```

```
for i in range(r):
    for j in range(sample_size):
        # Sample climatic realizations and categorize climatic realizations
        if list_sobol_points[i][j,0] < 1/15:
          np_sobol_stochastic[j,0] = 1
          np_sobol_stochastic_categories[j,0] = 'R01'
        elif list_sobol_points[i][j,0] >= 1/15 and list_sobol_points[i][j,0] < 2/15:
            np_sobol_stochastic[j,0] = 2
            np_sobol_stochastic_categories[j,0] = 'R02'
        elif list_sobol_points[i][j,0] >= 2/15 and list_sobol_points[i][j,0] < 3/15:
            np_sobol_stochastic[j,0] = 3
            np_sobol_stochastic_categories[j,0] = 'R03'
        elif list_sobol_points[i][j,0] >= 3/15 and list_sobol_points[i][j,0] < 4/15:
            np_sobol_stochastic[j,0] = 4
            np_sobol_stochastic_categories[j,0] = 'R04'
        elif list_sobol_points[i][j,0] >= 4/15 and list_sobol_points[i][j,0] < 5/15:
            np_sobol_stochastic[j,0] = 5
            np_sobol_stochastic_categories[j,0] = 'R05'
        elif list_sobol_points[i][j,0] >= 5/15 and list_sobol_points[i][j,0] < 6/15:
            np_sobol_stochastic[j,0] = 6
            np_sobol_stochastic_categories[j,0] = 'R06'
        elif list_sobol_points[i][j,0] >= 6/15 and list_sobol_points[i][j,0] < 7/15:
            np_sobol_stochastic[j,0] = 7
            np_sobol_stochastic_categories[j,0] = 'R07'
        elif list_sobol_points[i][j,0] >= 7/15 and list_sobol_points[i][j,0] < 8/15:
            np_sobol_stochastic[j,0] = 8
            np_sobol_stochastic_categories[j,0] = 'R08'
        elif list_sobol_points[i][j,0] >= 8/15 and list_sobol_points[i][j,0] < 9/15:
            np_sobol_stochastic[j,0] = 9
            np_sobol_stochastic_categories[j,0] = 'R09'
        elif list_sobol_points[i][j,0] >= 9/15 and list_sobol_points[i][j,0] < 10/15:
            np_sobol_stochastic[j,0] = 10
            np_sobol_stochastic_categories[j,0] = 'R10'
        elif list_sobol_points[i][j,0] >= 10/15 and list_sobol_points[i][j,0] < 11/15:
            np_sobol_stochastic[j,0] = 11
            np_sobol_stochastic_categories[j,0] = 'R11'
        elif list_sobol_points[i][j,0] >= 11/15 and list_sobol_points[i][j,0] < 12/15:
            np_sobol_stochastic[j,0] = 12
            np_sobol_stochastic_categories[j,0] = 'R12'
```

```
        elif list_sobol_points[i][j,0] >= 12/15 and list_sobol_points[i][j,0] < 13/15:
            np_sobol_stochastic[j,0] = 13
            np_sobol_stochastic_categories[j,0] = 'R13'
        elif list_sobol_points[i][j,0] >= 13/15 and list_sobol_points[i][j,0] < 14/15:
            np_sobol_stochastic[j,0] = 14
            np_sobol_stochastic_categories[j,0] = 'R14'
        elif list_sobol_points[i][j,0] >= 14/15 and list_sobol_points[i][j,0] < 15/15:
            np_sobol_stochastic[j,0] = 15
            np_sobol_stochastic_categories[j,0] = 'R15'

        # Sample orientations
        np_sobol_stochastic[j,1] = np.percentile(np_stochastic_variables_continuous[:,0],100*list_sobol_points[i][j,1])
        # Categorize orientations
        if np_sobol_stochastic[j,1] > 348.75 or np_sobol_stochastic[j,1] < 11.25:
            np_sobol_stochastic_categories [j,1] = 'N'
        elif np_sobol_stochastic[j,1] >= 11.25 and np_sobol_stochastic[j,1] < 33.75:
            np_sobol_stochastic_categories [j,1] = 'NNE'
        elif np_sobol_stochastic[j,1] >= 33.75 and np_sobol_stochastic[j,1] < 56.25:
            np_sobol_stochastic_categories [j,1] = 'NE'
        elif np_sobol_stochastic[j,1] >= 56.25 and np_sobol_stochastic[j,1] < 78.75:
            np_sobol_stochastic_categories [j,1] = 'ENE'
        elif np_sobol_stochastic[j,1] >= 78.75 and np_sobol_stochastic[j,1] < 101.25:
            np_sobol_stochastic_categories [j,1] = 'E'
        elif np_sobol_stochastic[j,1] >= 101.25 and np_sobol_stochastic[j,1] < 123.75:
            np_sobol_stochastic_categories [j,1] = 'ESE'
        elif np_sobol_stochastic[j,1] >= 123.75 and np_sobol_stochastic[j,1] < 146.25:
            np_sobol_stochastic_categories [j,1] = 'SE'
        elif np_sobol_stochastic[j,1] >= 146.25 and np_sobol_stochastic[j,1] < 168.75:
            np_sobol_stochastic_categories [j,1] = 'SSE'
        elif np_sobol_stochastic[j,1] >= 168.75 and np_sobol_stochastic[j,1] < 191.25:
            np_sobol_stochastic_categories [j,1] = 'S'
        elif np_sobol_stochastic[j,1] >= 191.25 and np_sobol_stochastic[j,1] < 213.75:
            np_sobol_stochastic_categories [j,1] = 'SSW'
        elif np_sobol_stochastic[j,1] >= 213.75 and np_sobol_stochastic[j,1] < 236.25:
            np_sobol_stochastic_categories [j,1] = 'SW'
        elif np_sobol_stochastic[j,1] >= 236.25 and np_sobol_stochastic[j,1] < 258.75:
            np_sobol_stochastic_categories [j,1] = 'WSW'
        elif np_sobol_stochastic[j,1] >= 258.75 and np_sobol_stochastic[j,1] < 281.25:
            np_sobol_stochastic_categories [j,1] = 'W'
        elif np_sobol_stochastic[j,1] >= 281.25 and np_sobol_stochastic[j,1] < 303.75:
            np_sobol_stochastic_categories [j,1] = 'WNW'
```

```
        elif np_sobol_stochastic[j,1] >= 303.75 and np_sobol_stochastic[j,1] < 326.25:
            np_sobol_stochastic_categories [j,1] = 'NW'
        elif np_sobol_stochastic[j,1] >= 326.25 and np_sobol_stochastic[j,1] < 348.75:
            np_sobol_stochastic_categories [j,1] = 'NNW'

        # Sample rain deposition factor
        np_sobol_stochastic[j,2] = np.percentile(np_stochastic_variables_continuous[:,1],100*list_sobol_points[i][j,2])
        # Categorize rain deposition factor
        if np_sobol_stochastic [j,2] >= 0.35 and np_sobol_stochastic [j,2] < 0.56:
            np_sobol_stochastic_categories[j,2] = 'Low'
        elif np_sobol_stochastic [j,2] >= 0.56 and np_sobol_stochastic [j,2] < 0.78:
            np_sobol_stochastic_categories[j,2] = 'Middle'
        elif np_sobol_stochastic [j,2] >= 0.78 and np_sobol_stochastic [j,2] < 1:
            np_sobol_stochastic_categories[j,2] = 'High'

        # Sample rain leakage moisture source, and categorize the rain leakage moisture source
        if list_sobol_points[i][j,3] < 1/3:
          np_sobol_stochastic[j,3] = np.percentile(np_stochastic_variables_continuous[:,2],
                                                   100*(list_sobol_points[i][j,3]-0)/(1/3-0))
          np_sobol_stochastic_categories[j,3] = 'Low'
        elif list_sobol_points[i][j,3] >= 1/3 and list_sobol_points[i][j,3] < 2/3:
          np_sobol_stochastic[j,3] = np.percentile(np_stochastic_variables_continuous[:,3],
                                                   100*(list_sobol_points[i][j,3]-1/3)/(2/3-1/3))
          np_sobol_stochastic_categories[j,3] = 'Middle'
        elif list_sobol_points[i][j,3] >= 2/3 and list_sobol_points[i][j,3] < 1:
          np_sobol_stochastic[j,3] = np.percentile(np_stochastic_variables_continuous[:,4],
                                                   100*(list_sobol_points[i][j,3]-2/3)/(1-1/3))
          np_sobol_stochastic_categories[j,3] = 'High'

        # Sample ACH, and categorize the ACH
        if list_sobol_points[i][j,4] < 1/3:
          np_sobol_stochastic[j,4] = np.percentile(np_stochastic_variables_continuous[:,5],
                                                   100*(list_sobol_points[i][j,4]-0)/(1/3-0))
          np_sobol_stochastic_categories[j,4] = 'Low'
        elif list_sobol_points[i][j,4] >= 1/3 and list_sobol_points[i][j,4] < 2/3:
          np_sobol_stochastic[j,4] = np.percentile(np_stochastic_variables_continuous[:,6],
                                                   100*(list_sobol_points[i][j,4]-1/3)/(2/3-1/3))
          np_sobol_stochastic_categories[j,4] = 'Middle'
        elif list_sobol_points[i][j,4] >= 2/3 and list_sobol_points[i][j,4] < 1:
          np_sobol_stochastic[j,4] = np.percentile(np_stochastic_variables_continuous[:,7],
                                                   100*(list_sobol_points[i][j,4]-2/3)/(1-1/3))
          np_sobol_stochastic_categories[j,4] = 'High'

        # Sample effective saturation water content and A-value of brick based on the sampled indices
        np_sobol_stochastic [j,5] = np_stochastic_variables_continuous[list_sampled_A_Theta_index[i][j],8]
        np_sobol_stochastic [j,6] = np_stochastic_variables_continuous[list_sampled_A_Theta_index[i][j],9]

    # Store different sets of sobol stochastic variables, and empty the temprory sobol variables for next round sampling
    list_sobol_stochastic.append(np_sobol_stochastic)
    list_sobol_stochastic_categories.append(np_sobol_stochastic_categories)
    np_sobol_stochastic = np.empty([sample_size,7])
    np_sobol_stochastic_categories = np.empty([sample_size,5], dtype = 'object')

# Save the sampled stochastic sobol variables, and the categories of stochastic sobol variables
directory_stochastic_sobol_variables = \
    'D:/StochasticModelling/StochasticScripts/stochastic_sobol_variables/stochastic_sobol_variables_'
list_sobol_stochastic_variable_names = ['Climate','Orientation','FD','MS','ACH','A-value_Brick','Theta_Brick']
for i in range(r):
    df_sobol_stochastic_variables = pd.DataFrame (data=list_sobol_stochastic[i], columns = list_sobol_stochastic_variable_names)
    df_sobol_stochastic_variables.to_csv(directory_stochastic_sobol_variables + str(i) + '.csv')

directory_stochastic_sobol_variables_categories = \
    'D:/StochasticModelling/StochasticScripts/stochastic_sobol_variables/stochastic_sobol_variables_categories_'
list_sobol_stochastic_variable_categories_names = ['Climate','Orientation','FD','MS','ACH']
for i in range(r):
    df_sobol_stochastic_variables = pd.DataFrame (data=list_sobol_stochastic_categories[i],
                                                  columns = list_sobol_stochastic_variable_categories_names)
    df_sobol_stochastic_variables.to_csv(directory_stochastic_sobol_variables_categories  + str(i) + '.csv')
```

Appendix D Scripts for varying material properties

```python
import re
import math

def VaryMaterialProperties_BasicParameters (ProjectContent, Material, Parameter, Coefficient):

    ProjectContent_New = ProjectContent

    # Extract the indices of material and parameter
    Indices_Material = [i for i, s in enumerate(ProjectContent_New) if Material in s]
    Indices_Parameter = [i for i, s in enumerate(ProjectContent_New) if Parameter in s]


    # Identify the indice of the parameter within a specific material
    LineDiff_Material_Parameter = []
    for i in Indices_Parameter:
        LineDiff = i - Indices_Material[0]
        LineDiff_Material_Parameter.append(LineDiff)
    LineDiff_Material_Parameter_Min = min([i for i in LineDiff_Material_Parameter if i > 0])
    Indice_LineDiff_Material_Parameter_Min = LineDiff_Material_Parameter.index(LineDiff_Material_Parameter_Min)
    Indice_Material_Parameter = Indices_Parameter[Indice_LineDiff_Material_Parameter_Min]

    # Extract the parameter of a specific material
    Material_Parameter_Str = ProjectContent[ Indice_Material_Parameter ]
    Material_Parameter_Str_List = re.split(r'\s+', Material_Parameter_Str)
    Material_Parameter_Float = float(Material_Parameter_Str_List[3])

    # Vary the parameter by mutiplying a coefficient, and write the new parameter into a new ProjectContent list
    Material_Parameter_Str_New =  format( Material_Parameter_Float * Coefficient, '.7g')
    Material_Parameter_Replacement = Material_Parameter_Str.replace(Material_Parameter_Str_List[3],Material_Parameter_Str_New)
    ProjectContent_New[Indice_Material_Parameter] = Material_Parameter_Replacement
    return ProjectContent_New
```

```python

def VaryMaterialProperties_PropertyCurves (ProjectContent, Material, Parameter, Coefficient):

    ProjectContent_New = ProjectContent

    # Extract the indices of material and parameter
    Indices_Material = [i for i, s in enumerate(ProjectContent_New) if Material in s]
    Indices_Parameter = [i for i, s in enumerate(ProjectContent_New) if Parameter in s]

    # Identify the indice of the parameter within a specific material
    LineDiff_Material_Parameter = []
    for i in Indices_Parameter:
        LineDiff = i - Indices_Material[0]
        LineDiff_Material_Parameter.append(LineDiff)
    LineDiff_Material_Parameter_Min = min([i for i in LineDiff_Material_Parameter if i > 0])
    Indice_LineDiff_Material_Parameter_Min = LineDiff_Material_Parameter.index(LineDiff_Material_Parameter_Min)
    Indice_Material_Parameter = Indices_Parameter[Indice_LineDiff_Material_Parameter_Min] + 2

    # Extract the parameter of a specific material

    Material_Parameter_Str = ProjectContent[ Indice_Material_Parameter ]
    Material_Parameter_Str_List = re.findall('\d+\.?\d*',Material_Parameter_Str)
    if Parameter == 'Theta_l(RH)':
       Material_Parameter_Float = [float(i) for i in Material_Parameter_Str_List ]
    else :
       Material_Parameter_Float = [-float(i) for i in Material_Parameter_Str_List ]

    # Vary the parameter by mutiplying a coefficient, and write the new parameter into a new ProjectContent list
    if Parameter == 'Theta_l(RH)':
       Material_Parameter_Float_New = [i*Coefficient for i in Material_Parameter_Float]
       Material_Parameter_Str_New = [format(i,'.7g') for i in Material_Parameter_Float_New]
       Material_Parameter_Replacement = '          '
       for i in Material_Parameter_Str_New:
           Material_Parameter_Replacement = Material_Parameter_Replacement + i + '          '

    else :
       Material_Parameter_Float_New = [math.pow(10,i) * Coefficient for i in Material_Parameter_Float]
       Material_Parameter_Str_New = [format(math.log10(i),'.7g') for i in Material_Parameter_Float_New]
       Material_Parameter_Replacement = '          '
       for i in Material_Parameter_Str_New:
            Material_Parameter_Replacement = Material_Parameter_Replacement + i + '          '
```

```python


    ProjectContent_New[Indice_Material_Parameter] = Material_Parameter_Replacement

    # If the varied parameter is SC, then the SC that is used for scaling Dl should be varied also

    if Parameter == 'Theta_l(RH)':
        Indice_Material_Parameter_ScaleDl = Indices_Parameter[Indice_LineDiff_Material_Parameter_Min] + 7
        Indice_Material_Parameter_ScaleKv = Indices_Parameter[Indice_LineDiff_Material_Parameter_Min] + 10
        Indice_Material_Parameter_Scalelambda = Indices_Parameter[Indice_LineDiff_Material_Parameter_Min] + 16

        Material_Parameter_ScaleDl_Str = ProjectContent[ Indice_Material_Parameter_ScaleDl ]
        Material_Parameter_ScaleDl_Str_List = re.findall('\d+\.?\d*',Material_Parameter_ScaleDl_Str)
        Material_Parameter_ScaleKv_Str = ProjectContent[ Indice_Material_Parameter_ScaleKv ]
        Material_Parameter_ScaleKv_Str_List = re.findall('\d+\.?\d*',Material_Parameter_ScaleKv_Str)
        Material_Parameter_Scalelambda_Str = ProjectContent[ Indice_Material_Parameter_Scalelambda ]
        Material_Parameter_Scalelambda_Str_List = re.findall('\d+\.?\d*',Material_Parameter_Scalelambda_Str)
        Material_Parameter_ScaleDl_Float = [float(i) for i in Material_Parameter_ScaleDl_Str_List ]
        Material_Parameter_ScaleKv_Float = [float(i) for i in Material_Parameter_ScaleKv_Str_List ]
        Material_Parameter_Scalelambda_Float = [float(i) for i in Material_Parameter_Scalelambda_Str_List ]

        Material_Parameter_ScaleDl_Float_New = [i*Coefficient for i in Material_Parameter_ScaleDl_Float]
        Material_Parameter_ScaleDl_Str_New = [format(i,'.7g') for i in Material_Parameter_ScaleDl_Float_New]
        Material_Parameter_ScaleKv_Float_New = [i*Coefficient for i in Material_Parameter_ScaleKv_Float]
        Material_Parameter_ScaleKv_Str_New = [format(i,'.7g') for i in Material_Parameter_ScaleKv_Float_New]
        Material_Parameter_Scalelambda_Float_New = [i*Coefficient for i in Material_Parameter_Scalelambda_Float]
        Material_Parameter_Scalelambda_Str_New = [format(i,'.7g') for i in Material_Parameter_Scalelambda_Float_New]
        Material_Parameter_ScaleDl_Replacement = '          '
        for i in Material_Parameter_ScaleDl_Str_New:
            Material_Parameter_ScaleDl_Replacement = Material_Parameter_ScaleDl_Replacement + i + '          '


        Material_Parameter_ScaleKv_Replacement = '          '
        for i in Material_Parameter_ScaleKv_Str_New:
            Material_Parameter_ScaleKv_Replacement = Material_Parameter_ScaleKv_Replacement + i + '          '
        Material_Parameter_Scalelambda_Replacement = '          '
        for i in Material_Parameter_Scalelambda_Str_New:
            Material_Parameter_Scalelambda_Replacement = Material_Parameter_Scalelambda_Replacement + i + '          '

```

```python


        ProjectContent_New[Indice_Material_Parameter_ScaleDl] = Material_Parameter_ScaleDl_Replacement
        ProjectContent_New[Indice_Material_Parameter_ScaleKv] = Material_Parameter_ScaleKv_Replacement
        ProjectContent_New[Indice_Material_Parameter_Scalelambda] = Material_Parameter_Scalelambda_Replacement

    return ProjectContent_New


def Modify_ResultFolderName (ProjectContent, ProjectName, num_Project):
    ResultFolder_Project_Dir = 'OUTPUT_FOLDER            = $(PROJECT_DIR)/'
    Indice_ResultFolder = [i for i, s in enumerate(ProjectContent) if ResultFolder_Project_Dir in s]
    ProjectContent[Indice_ResultFolder[0]] = ResultFolder_Project_Dir + ProjectName + '_' + str(num_Project) + '.results'

    return ProjectContent
```

Appendix E Scripts for varying climate data

1. Functions for varying climate data

```
import pandas as pd
import math

def generating_ccd (ClimateData, list_Years, Wall_Orientation, F_D_E, RL_Percentage, RL_Thickness):

    ClimateData.set_index('YEAR', inplace = True)
    ClimateData_Grouped = ClimateData.groupby(['YEAR'])
    list_ClimateData_NewSequence = list()

    # Construct new climate sequence based on selected years
    for i in list_Years:
        ClimateData_Year = ClimateData_Grouped.get_group(i)
        list_ClimateData_NewSequence.append(ClimateData_Year)
    pd_ClimateData_NewSequence = pd.concat(list_ClimateData_NewSequence)

    # Extract climate parameters
    series_Temperature = pd_ClimateData_NewSequence['TEMP_DegC']
    list_Temperature = list()
    for i in range (len(series_Temperature)):
        str_Temperature = str(series_Temperature.iloc[i])
        list_Temperature.append(str_Temperature)

    series_RH = pd_ClimateData_NewSequence['RHUM_Percent']
    list_RH = list()
    for i in range(len(series_RH)):
        str_RH = str(series_RH.iloc[i])
        list_RH.append(str_RH)

    series_WDIR = pd_ClimateData_NewSequence['WDIR_ClockwiseDegFromNorth']
    list_WDIR = list()
    for i in range(len(series_WDIR)):
        str_WDIR = str(series_WDIR.iloc[i])
        list_WDIR.append(str_WDIR)

    series_WSP = pd_ClimateData_NewSequence['WSP_MPerSec']
    list_WSP = list()
    for i in range(len(series_WSP)):
        str_WSP = str(series_WSP.iloc[i])
        list_WSP.append(str_WSP)

    series_DRI = pd_ClimateData_NewSequence['DRI_kJperm2']
    list_DRI = list()
    for i in range(len(series_DRI)):
        str_DRI = str(series_DRI.iloc[i] / 3.6)
        list_DRI.append(str_DRI)

    series_DHI = pd_ClimateData_NewSequence['DHI_kJperm2']
    list_DHI = list()
    for i in range(len(series_DHI)):
        str_DHI = str(series_DHI.iloc[i] / 3.6)
        list_DHI.append(str_DHI)

    series_Cloudiness = pd_ClimateData_NewSequence['TCC_Percent']
    list_Cloudiness = list()
    for i in range(len(series_Cloudiness)):
        str_Cloudiness = str(series_Cloudiness.iloc[i] / 100)
        list_Cloudiness.append(str_Cloudiness)

    series_Horizontal_Rain = pd_ClimateData_NewSequence['RAIN_Mm']
    list_Horizontal_Rain =  list()
    for i in range(len(series_Horizontal_Rain)):
        str_Horizontal_Rain = str(series_Horizontal_Rain.iloc[i])
        list_Horizontal_Rain.append(str_Horizontal_Rain)


    series_WDR = calculate_wdr (series_Horizontal_Rain, series_WDIR, series_WSP, F_D_E, Wall_Orientation)
    list_WDR = list()
    for i in range(len(series_WDR)):
        str_WDR = str(series_WDR.iloc[i])
        list_WDR.append(str_WDR)
```

```
    series_RL = 1000 * RL_Percentage * series_WDR / (3600 * RL_Thickness)
    list_RL = list()
    for i in range(len(series_RL)):
        str_RL = str(series_RL.iloc[i])
        list_RL.append(str_RL)

    series_SkyEmission = calculate_skyemission (series_Temperature, series_RH, series_Cloudiness)
    list_SkyEmission = list()
    for i in range(len(series_SkyEmission)):
        str_SkyEmission = str(series_SkyEmission.iloc[i])
        list_SkyEmission.append(str_SkyEmission)

    series_GroundEmission = series_Temperature.apply(lambda t: (5.67E-8) * ((t+273.15) ** 4))
    list_GroundEmission = list()
    for i in range(len(series_GroundEmission)):
        str_GroundEmission = str(-series_GroundEmission.iloc[i])
        list_GroundEmission.append(str_GroundEmission)

    # Construct day column
    ClimateData_FirstYear = ClimateData_Grouped.get_group(list_Years[0])
    series_Day_FirstYear = ClimateData_FirstYear['YDAY'] - 1
    list_Day = list()
    for i in range (len(list_Years)):
        series_Day = series_Day_FirstYear + i * 365
        list_Day.append(series_Day)
    series_Day_NewSequence = pd.concat(list_Day)
    list_Day.clear()
    for i in range (len(series_Day_NewSequence)):
        str_Day = str(series_Day_NewSequence.iloc[i])
        list_Day.append(str_Day)

    # Construct hour column
    series_Hour = pd_ClimateData_NewSequence ['HOUR'] - 1
    list_Hour = list()
    for i in range(len(series_Hour)):
        str_hour = str(series_Hour.iloc[i]) + ':00:00'
        list_Hour.append(str_hour)

```

```
    # Construct list of ccd file
    list_ccd_Temperature = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_Temperature[i]
        list_ccd_Temperature.append(str_ccd)
    list_Header_Temperature = 'TEMPER C'
    list_ccd_Temperature.insert(0, list_Header_Temperature)

    list_ccd_RH = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_RH[i]
        list_ccd_RH.append(str_ccd)
    list_Header_RH = 'RELHUM %'
    list_ccd_RH.insert(0, list_Header_RH)

    list_ccd_WDIR = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_WDIR[i]
        list_ccd_WDIR.append(str_ccd)
    list_Header_WDIR = 'WINDDIR Deg'
    list_ccd_WDIR.insert(0, list_Header_WDIR)

    list_ccd_WSP = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_WSP[i]
        list_ccd_WSP.append(str_ccd)
    list_Header_WSP = 'WINDVEL m/s'
    list_ccd_WSP.insert(0, list_Header_WSP)

    list_ccd_DRI = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_DRI[i]
        list_ccd_DRI.append(str_ccd)
    list_Header_DRI = 'DIRRAD W/m2'
    list_ccd_DRI.insert(0, list_Header_DRI)

```

```
    list_ccd_DHI = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_DHI[i]
        list_ccd_DHI.append(str_ccd)
    list_Header_DHI = 'DIFRAD W/m2'
    list_ccd_DHI.insert(0, list_Header_DHI)

    list_ccd_Cloudiness = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_Cloudiness[i]
        list_ccd_Cloudiness.append(str_ccd)
    list_Header_Cloudiness = 'CLOUDCOV -'
    list_ccd_Cloudiness.insert(0, list_Header_Cloudiness)

    list_ccd_WDR = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_WDR[i]
        list_ccd_WDR.append(str_ccd)
    list_Header_WDR = 'NORRAIN l/m2h'
    list_ccd_WDR.insert(0, list_Header_WDR)

    list_ccd_RL = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_RL[i]
        list_ccd_RL.append(str_ccd)
    list_Header_RL = 'WATSOURCE kg/m3s'
    list_ccd_RL.insert(0, list_Header_RL)

    list_ccd_SkyEmission = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_SkyEmission[i]
        list_ccd_SkyEmission.append(str_ccd)
    list_Header_SkyEmission = 'SKYEMISS W/m2'
    list_ccd_SkyEmission.insert(0, list_Header_SkyEmission)

```

```
    list_ccd_GroundEmission = list()
    for i in range(len(series_Hour)):
        str_ccd = list_Day[i] + ' ' + list_Hour[i] + ' ' + list_GroundEmission[i]
        list_ccd_GroundEmission.append(str_ccd)
    list_Header_GroundEmission = 'GRNDEMISS W/m2'
    list_ccd_GroundEmission.insert(0, list_Header_GroundEmission)




    return list_ccd_Temperature, list_ccd_RH, list_ccd_WDIR, list_ccd_WSP, list_ccd_DRI, list_ccd_DHI, list_ccd_Cloudiness, \
           list_ccd_WDR, list_ccd_RL, list_ccd_SkyEmission, list_ccd_GroundEmission

def save_ccd (Directory_ccd, list_ccd, name_ccd):
    ccd_file = open (Directory_ccd + name_ccd + '.ccd' , 'w', encoding = 'utf-8')
    for i in list_ccd:
        ccd_file.write("%s\n" % i)
    ccd_file.close()

def calculate_wdr (series_Horizontal_Rain, series_WDIR, series_WSP, F_D_E, Wall_Orientation):
    cosine_normal  = calculate_cosine_normal (Wall_Orientation, series_WDIR)
    RainFactor = 0.2 * F_D_E
    wdr = RainFactor * series_Horizontal_Rain * series_WSP * cosine_normal
    return wdr

def calculate_cosine_normal(wall_orientation, wind_direction):
    # Compute angle between normal to the wall and wind direction
    angle_normal_series = wind_direction.apply(
        lambda wind_dir: abs(wall_orientation - wind_dir))
    angle_normal_series = angle_normal_series.apply(lambda ang: 360 - ang if ang > 270 else ang)
    angle_normal_series = angle_normal_series.apply(
        lambda ang: min(90, ang))  # If less than 90, replace value
    angle_normal_series = angle_normal_series.apply(lambda ang: ang * math.pi / 180)
    cosine_normal = angle_normal_series.apply(lambda ang: max(0, math.cos(ang)))

    return cosine_normal

```

```python
def calculate_skyemission (temperature, RH, cloudiness):
    # Calculate saturated pressure based on equation from Hugo Hens, Fundamentals
    saturated_pressure_series = temperature.apply(lambda temp: calculate_saturation_vp(temp))


    # Calculate vapour pressure, relative humidity must be converted from percent
    vapour_pressure = saturated_pressure_series * RH / 100

    # Calculate Sky Temperature
    f_rh = vapour_pressure.apply(lambda vp: 0.82 - 0.25 * 10 ** (-0.001333224 * vp))
    f_cc_perc = cloudiness.apply(lambda cc: 1 + 0.2 * (cc/100) ** 2)
    f_rh_cc = f_rh * f_cc_perc
    # Below Calculates: temp_kelvin_series * (f_rh * f_cc_perc) ** (1 / 4)
    list_t_sky_coefficient = list()

    for i in range(len(temperature)):
        t_sky_coefficient =  math.pow (f_rh_cc.iloc[i], 0.25)
        list_t_sky_coefficient.append(t_sky_coefficient)

    series_t_sky_coefficient = pd.Series(data=list_t_sky_coefficient, dtype=float)


    series_temperature_k = temperature + 273.15

    list_sky_temperature = list()
    for i in range(len(temperature)):
        sky_temperature = series_temperature_k.iloc[i] * series_t_sky_coefficient.iloc[i]
        list_sky_temperature.append(sky_temperature)
    series_sky_temperature = pd.Series(data = list_sky_temperature, dtype = float )

    # Convert sky_temperature to Celsius
    # t_sky_perc_c = t_sky_perc_k.sub(273.15)

    sky_emission = series_sky_temperature.apply(lambda t: (5.67E-8) * (t ** 4))

    return sky_emission

```

```python
def calculate_saturation_vp (temp_c):
    const = [-5.6745359e3, 6.3925247, -9.677843e-3, 6.22115701e-7, 2.0747825e-9, -9.484024e-13, 4.1635019,
                 -5.80022006e3,
                 1.3914993, -4.8640239e-2, 4.1764768e-5, -1.4452093e-8, 6.5459673]
        # Temperature in kelvin
    temp_k = temp_c + 273.15

        # Formula from "2009 ASHRAE Handbook - Fundamentals"
    if temp_c < 0:
        ln_saturation_vapour_pressure = const[0] / temp_k + const[1] + const[2] * temp_k + const[3] * (
                     temp_k ** 2) + \
                                            const[4] * (temp_k ** 3) + const[5] * (temp_k ** 4) + const[
                                                6] * math.log(
                 temp_k)
    else:
        ln_saturation_vapour_pressure = const[7] / temp_k + const[8] + const[9] * temp_k + const[10] * (
                     temp_k ** 2) + \
                                            const[11] * (temp_k ** 3) + const[12] * math.log(temp_k)

        # Return saturation vapour pressure e^ln(saturation_vapour_pressure)
    saturation_vapour_pressure = math.exp(ln_saturation_vapour_pressure)
    return saturation_vapour_pressure
```

2. Main routine for generating stochastic CCD files

```python
import pandas as pd
from ReadClimateData import ReadClimateFile
from GeneratingCCDFile_MixedRainFactor import generating_ccd
from GeneratingCCDFile_MixedRainFactor import save_ccd
import os


# Import the random numbers of the stochastic variables
list_Stochastic_Variables = list()
for i in range(10):
    pd_Stochastic_Variables = \
        pd.read_csv(
            r'D:\StochasticModelling\Building_Special_Issue\Stochastic_Inputs\Stochastic_Inputs_256\Stochastic_Variables_Simulation_256
                + str(i+1) + '.csv' , sep = ',')
    list_Stochastic_Variables.append(pd_Stochastic_Variables)



# Specify the directory of the climate data, the selected Moisture Reference Years based on Moisture Index
Directory_ClimateData = 'D:/StochasticModelling/ClimateData/'
Directory_2Years_MRYs = 'D:/StochasticModelling/ClimateData/MI/'
Directory_CCDFiles_F0 = 'D:/StochasticModelling/Building_Special_Issue/CCDFiles/F0/'
Directory_CCDFiles_F7 = 'D:/StochasticModelling/Building_Special_Issue/CCDFiles/F7/'

# Read the climate data, and save it to a list containing the climate data of all the 15 realizations
ClimateData_Ott_F0 = ReadClimateFile(Directory_ClimateData,'Ottawa', 'F0', 'Ott')
ClimateData_Ott_F7 = ReadClimateFile(Directory_ClimateData,'Ottawa', 'F7', 'Ott')

# Read the Moisture Reference Years
MRY_2Years_Ott_F0 = pd.read_csv(Directory_2Years_MRYs + 'Ottawa/MRY_AW_Ott_F0.csv')
MRY_2Years_Ott_F7 = pd.read_csv(Directory_2Years_MRYs + 'Ottawa/MRY_AW_Ott_F7.csv')
MRY_2Years_Ott_F0.set_index('Unnamed: 0', inplace=True)
MRY_2Years_Ott_F7.set_index('Unnamed: 0', inplace=True)
```

```python

# Create a list of the years, whose climate data will be generated
list_df_MRY_2Years_Ott = list()
list_MRY_2Years_Ott_F0 = list()
list_MRY_2Years_Ott_F7 = list()

list_df_MRY_2Years_Ott.append(MRY_2Years_Ott_F0)
list_df_MRY_2Years_Ott.append(MRY_2Years_Ott_F7)

for i in range(15):
    list_MRY = list_df_MRY_2Years_Ott[0].iloc[:,i].to_list()
    list_MRY_2Years_Ott_F0.append(list_MRY)
    list_MRY = list_df_MRY_2Years_Ott[1].iloc[:,i].to_list()
    list_MRY_2Years_Ott_F7.append(list_MRY)

list_ccd_name = ['Temperature','RelativeHumidity','WindDirection','WindSpeed','DirectSolarRadiation','DiffuseSolarRadiation',
                 'Cloudiness','WDR','WaterSource','SkyRadiation','GroundEmission']

# Generate CCDFiles and save them to specified locations
for i in range (10):
    # os.makedirs(Directory_CCDFiles_F0 +  'Randomized_' + str(i) )
    for j in range (192,256):
        list_ccd_Ott_F0 = generating_ccd(ClimateData_Ott_F0[list_Stochastic_Variables[i].iloc[j,0]-1].copy(),
                                         list_MRY_2Years_Ott_F0[list_Stochastic_Variables[i].iloc[j,0]-1],
                                         list_Stochastic_Variables[i].iloc[j,1], list_Stochastic_Variables[i].iloc[j,2],
                                         list_Stochastic_Variables[i].iloc[j,3]/100, 0.08 )
        os.makedirs(Directory_CCDFiles_F0 +  'Randomized_' + str(i) + '/CCD_' + str(j) )
        for k in range (11):
            save_ccd(Directory_CCDFiles_F0 + 'Randomized_' + str(i) + '/CCD_' + str(j) + '/' , list_ccd_Ott_F0[k], list_ccd_name[k])

for i in range (10):
    # os.makedirs(Directory_CCDFiles_F7 +  'Randomized_' + str(i) )
    for j in range (192,256):
        list_ccd_Ott_F7 = generating_ccd(ClimateData_Ott_F7[list_Stochastic_Variables[i].iloc[j,0]-1].copy(),
                                         list_MRY_2Years_Ott_F7[list_Stochastic_Variables[i].iloc[j,0]-1],
                                         list_Stochastic_Variables[i].iloc[j,1], list_Stochastic_Variables[i].iloc[j,2],
                                         list_Stochastic_Variables[i].iloc[j,3]/100, 0.08 )
        os.makedirs(Directory_CCDFiles_F7 +  'Randomized_' + str(i) + '/CCD_' + str(j) )
        for k in range (11):
            save_ccd(Directory_CCDFiles_F7 + 'Randomized_' + str(i) + '/CCD_' + str(j) + '/' , list_ccd_Ott_F7[k], list_ccd_name[k])
```

Appendix F Scripts for generating stochastic dpj files

```python
from VaryMaterialParameters import VaryMaterialProperties_BasicParameters
from VaryMaterialParameters import VaryMaterialProperties_PropertyCurves
import pandas as pd

filename_base = r'D:\StochasticModelling\Building_Special_Issue\For Jimmy\OTT_H_Base.dpj'

# Open DELPHIN project file and read each line into a list

File_BaseProject=open(filename_base,'rt',encoding = 'utf-8')
ProjectName_Base = 'OTT_H_Base'
ProjectContent_Base=File_BaseProject.readlines()
File_BaseProject.close()
ProjectContent_New = ProjectContent_Base # Define a new list that will include the varied parameters
list_ProjectContent_New = list()

# Read stochastic variables, in this case, there are 10 sets of 7 stochastic variables, each variable has 256 random numbers
list_Stochastic_Variables = list()

for i in range(10):
    pd_Stochastic_Variables = pd.read_csv\
        ('D:/StochasticModelling/Building_Special_Issue/For Jimmy/Stochastic_Inputs_256/Stochastic_Variables_Simulation_256_' +
         str(i+1) +'.csv', sep = ',')
    list_Stochastic_Variables.append(pd_Stochastic_Variables)

mu_Brick_LD = 6.18E-8
mu_Brick_SC = 0.2167

# Calculate stochastic coefficient of material properties
list_Stochastic_Coefficient_Brick_LD = list()
list_Stochastic_Coefficient_Brick_SC = list()
for i in range(10):
    pd_Stochastic_Coefficient_Brick_LD = list_Stochastic_Variables[i].iloc[:,5] / mu_Brick_LD
    pd_Stochastic_Coefficient_Brick_SC = list_Stochastic_Variables[i].iloc[:,6] / (mu_Brick_SC*1000)
    list_Stochastic_Coefficient_Brick_LD.append(pd_Stochastic_Coefficient_Brick_LD)
    list_Stochastic_Coefficient_Brick_SC.append(pd_Stochastic_Coefficient_Brick_SC)

#   Extract base ACH
ACH_base = 'CONSTVALUE                = 5 1/h'
ACH_Indice = [j for j, s in enumerate(ProjectContent_New) if ACH_base in s]

# Extract in indice of orientation
Orientation_base = 'ORIENTATION               = 258.3 Deg'
Orientation_Indice = [j for j, s in enumerate(ProjectContent_New) if Orientation_base in s]

# Extract the indece of result folder
ResultFolder_base = 'OUTPUT_FOLDER            = $(PROJECT_DIR)/Ott_H_Base.results'
ResultFolder_Indice = [j for j, s in enumerate(ProjectContent_New) if ResultFolder_base in s]

# Extract the indices of ccd files
list_ccdfile_names = ['Temperature.ccd',  'RelativeHumidity.ccd', 'WindDirection.ccd', 'WindSpeed.ccd', 'DiffuseSolarRadiation.ccd',
                      'DirectSolarRadiation.ccd', 'SkyRadiation.ccd', 'WDR.ccd','WaterSource.ccd','GroundEmission.ccd']
list_CCDDir_Indices = list()
for i in list_ccdfile_names:
    CCDDir_Indices = [j for j, s in enumerate(ProjectContent_New) if i in s]
    list_CCDDir_Indices.append(CCDDir_Indices[0])

# Extract the directory of ccd files
list_CCDDirectory = list()
for i in list_CCDDir_Indices:
    CCDDirectory = ProjectContent_New[i]
    list_CCDDirectory.append(CCDDirectory)
```

```python

# Generate stochastic dpj files
for i in range(10):

    for j in range(192,256):
        # Vary basic material parameters
        ProjectContent_New = VaryMaterialProperties_BasicParameters (ProjectContent_Base.copy(),'Brick II - Red Matt Clay Brick',
                                                                     'THETA_POR', list_Stochastic_Coefficient_Brick_SC[i].iloc[j])
        ProjectContent_New = VaryMaterialProperties_BasicParameters (ProjectContent_New,'Brick II - Red Matt Clay Brick','THETA_EFF',
                                                                     list_Stochastic_Coefficient_Brick_SC[i].iloc[j])
        ProjectContent_New = VaryMaterialProperties_BasicParameters (ProjectContent_New,'Brick II - Red Matt Clay Brick','THETA_CAP',
                                                                     list_Stochastic_Coefficient_Brick_SC[i].iloc[j])
        ProjectContent_New = VaryMaterialProperties_BasicParameters (ProjectContent_New,'Brick II - Red Matt Clay Brick','DLEFF',
                                                                     list_Stochastic_Coefficient_Brick_LD[i].iloc[j])

        # Vary material property curves
        ProjectContent_New = VaryMaterialProperties_PropertyCurves (ProjectContent_New,'Brick II - Red Matt Clay Brick',
                                                                    'Theta_l(RH)', list_Stochastic_Coefficient_Brick_SC[i].iloc[j])
        ProjectContent_New = VaryMaterialProperties_PropertyCurves (ProjectContent_New,'Brick II - Red Matt Clay Brick',
                                                                    'lgDl(Theta_l)', list_Stochastic_Coefficient_Brick_LD[i].iloc[j])

        # Vary ACH
        ProjectContent_New[ACH_Indice[0]] = ACH_base.replace('5' , str(list_Stochastic_Variables[i].iloc[j,4]))
        # Vary orientation
        ProjectContent_New[Orientation_Indice[0]] = Orientation_base.replace('258.3', str(list_Stochastic_Variables[i].iloc[j,1]))

        # Vary climate data
        for k in range(10):
            new_CCDDirectory = list_CCDDirectory[k].replace('Randomized_0',  'Randomized_' + str(i))
            new_CCDDirectory = new_CCDDirectory.replace('CCD_0',  'CCD_' + str(j))
            ProjectContent_New[list_CCDDir_Indices[k]] = new_CCDDirectory

        # Vary result folder
        ProjectContent_New[ResultFolder_Indice[0]] = ResultFolder_base.replace('Base',str(j))

        #     # Write new project file
        File_NewProject = open('D:/StochasticModelling/Building_Special_Issue/For Jimmy/Stochastic_Folders/Randomized_'  +   str(i) +
                               '/' + 'Ott_H_' + str(j) + '.dpj' , 'w', encoding = 'utf-8')
        for item in ProjectContent_New:
            File_NewProject.write("%s\n" % item)
        File_NewProject.close()
        ProjectContent_New = ProjectContent_Base

```

Appendix G Scripts for launching stochastic simulation

1. Function for calling the DELPHIN solver

```
import subprocess
import threading, queue

delphin_executable = r'C:/Program Files (x86)/IBK/Delphin 5.9/delphin_solver.exe'

q = queue.Queue()

def worker():

    while True:
        item = q.get()
        print('Running' + ' ' + item)
        retcode = subprocess . call ([ delphin_executable , "-x", "-v1", item ])
        if retcode != 0:
            print('Calculation error in' + ' ' + item + '!!!')
        else:
            print ('Calculation in' + ' ' + item + ' successfully completed')
        q. task_done ()
```

2. Main routine for launching stochastic simulation

```
for i in range(10):

# # turn-on the worker thread
    for j in range (32):
        threading.Thread(target=worker, daemon=True).start()

# send  task requests to the worker
    for j in range(32):
        q. put ('D:/StochasticModelling/Building_Special_Issue/Stochastic_Models/F0/'  + 'Randomized_' + str(i) + '/' + 'Ott_H_' +
                str(j+192) + '.dpj')

# block until all tasks are done
    q.join()
    print('Parallel ' + str(i) + ' completed')

for i in range(10):

# # turn-on the worker thread
    for j in range (32):
        threading.Thread(target=worker, daemon=True).start()

# send  task requests to the worker
    for j in range(32):
        q. put ('D:/StochasticModelling/Building_Special_Issue/Stochastic_Models/F0/'  + 'Randomized_' + str(i) + '/' + 'Ott_H_' +
                str(j+224) + '.dpj')

# block until all tasks are done
    q.join()
    print('Parallel ' + str(i) + ' completed')
```

Appendix H Scripts for error estimation

1. Functions of error estimation

```python
import numpy as np
import math

# This function estimates the standard error of the whole sample space
# The inputs are : r the number of sets of randomized sobol sequence, n the sample size for each set,
# np_MoldIndex a numpy array of mould growth index with size of n*r
def Estimate_Error (r,n,np_MoldIndex):

    np_MoldIndex_n = np.empty([n,r])
    np_MoldIndex_n = np.empty([n,r])

    for i in range (r):
        for j in range(n):
            np_MoldIndex_n[j,i] = np_MoldIndex[j,i]

    np_MoldIndex_R_mean = np.mean(np_MoldIndex_n, axis = 0)
    np_MoldIndex_R_std = np.std(np_MoldIndex_n, axis = 0)

    np_MoldIndex_E_mean = np.mean(np_MoldIndex_R_mean)
    np_MoldIndex_E_std = np.mean(np_MoldIndex_R_std)

    r_Coefficient = 1/(r*(r-1))

    np_diff_E_R_mean = np.empty([r,1])
    np_diff_E_R_std = np.empty([r,1])

    for i in range(r):
        np_diff_E_R_mean[i] = math.pow(np_MoldIndex_R_mean[i]-np_MoldIndex_E_mean , 2)
    np_sum_diff_E_R_mean = sum(np_diff_E_R_mean)
    stderr_mean = math.pow(r_Coefficient * np_sum_diff_E_R_mean,0.5)

    for i in range(r):
        np_diff_E_R_std[i] = math.pow(np_MoldIndex_R_std[i]-np_MoldIndex_E_std , 2)
    np_sum_diff_E_R_std = sum(np_diff_E_R_std)
    stderr_std = math.pow(r_Coefficient * np_sum_diff_E_R_std,0.5)

    return np_MoldIndex_E_mean,np_MoldIndex_E_std,stderr_mean,stderr_std
```

```python

# This function estimates the standard error of different orientations
# The inputs are : r the number of sets of randomized sobol sequence, n the sample size for each set
# np_MoldIndex a numpy array of mould growth index with size of nxr,
# np_Orientation a numpy array of the orientation for the stochastic cases with size of nxr
def Estimate_Error_Orientation (r,n,np_MoldIndex, np_Orientation):
    r_Coefficient = 1/(r*(r-1))
    list_Orientation = ['N', 'NNE','NE','ENE','E','ESE','SE','SSE','S','SSW','SW','WSW','W','WNW','NW','NNW']
    list_Group_Orientation = list()
    for i in list_Orientation:
        list_Index_Orientation = Group_Orientation (np_Orientation, i, n)
        list_Group_Orientation.append(list_Index_Orientation)

    np_MouldIndex_Orientation_R_mean = np.empty([10])
    np_MouldIndex_Orientation_R_std = np.empty([10])

    np_MouldIndex_Orientation_E_mean = np.empty([16])
    np_MouldIndex_Orientation_E_std = np.empty([16])

    np_MouldIndex_Orientation_diff_E_R_mean = np.empty([10])
    np_MouldIndex_Orientation_diff_E_R_std = np.empty([10])

    np_MoldIndex_Orientation_sum_diff_E_R_mean = np.empty([16])
    np_MoldIndex_Orientation_sum_diff_E_R_std = np.empty([16])

    np_stderr_Orientation_mean = np.empty([16])
    np_stderr_Orientation_std = np.empty([16])

    list_MoldIndex_Orientation_R_mean = list()
    list_MoldIndex_Orientation_R_std = list()

    list_MouldIndex_Orientation_diff_E_R_mean = list()
    list_MouldIndex_Orientation_diff_E_R_std = list()
```

```python

    for k in range(16):
        for i in range(r):
            np_MouldIndex_Orientation = np.empty([len(list_Group_Orientation[k][i])])
            for j in range(len(list_Group_Orientation[k][i])):
                np_MouldIndex_Orientation[j] = np_MoldIndex [list_Group_Orientation[k][i][j],i]
            np_MouldIndex_Orientation_R_mean[i] = np.mean(np_MouldIndex_Orientation, axis = 0)
            np_MouldIndex_Orientation_R_std[i] = np.std(np_MouldIndex_Orientation, axis = 0)

        list_MoldIndex_Orientation_R_mean.append(np_MouldIndex_Orientation_R_mean)
        list_MoldIndex_Orientation_R_std.append(np_MouldIndex_Orientation_R_std)
        np_MouldIndex_Orientation_E_mean[k] = np.mean(np_MouldIndex_Orientation_R_mean)
        np_MouldIndex_Orientation_E_std[k] = np.mean(np_MouldIndex_Orientation_R_std)

        for i in range(r):

            np_MouldIndex_Orientation_diff_E_R_mean[i] = math.pow(np_MouldIndex_Orientation_R_mean[i] -
                                                                  np_MouldIndex_Orientation_E_mean[k],2)
            np_MouldIndex_Orientation_diff_E_R_std[i] = math.pow(np_MouldIndex_Orientation_R_std[i] -
                                                                 np_MouldIndex_Orientation_E_std[k],2)

        list_MouldIndex_Orientation_diff_E_R_mean.append(np_MouldIndex_Orientation_diff_E_R_mean)
        list_MouldIndex_Orientation_diff_E_R_std.append(np_MouldIndex_Orientation_diff_E_R_std)

        np_MoldIndex_Orientation_sum_diff_E_R_mean[k] = sum (np_MouldIndex_Orientation_diff_E_R_mean)
        np_MoldIndex_Orientation_sum_diff_E_R_std[k] = sum (np_MouldIndex_Orientation_diff_E_R_std)

        np_stderr_Orientation_mean[k] = math.pow(r_Coefficient * np_MoldIndex_Orientation_sum_diff_E_R_mean[k] , 0.5)
        np_stderr_Orientation_std[k] = math.pow(r_Coefficient * np_MoldIndex_Orientation_sum_diff_E_R_std[k] , 0.5)

    return np_MouldIndex_Orientation_E_mean, np_MouldIndex_Orientation_E_std, np_stderr_Orientation_mean,np_stderr_Orientation_std

# This function estimates the standard error of different climatic realizations
# The inputs are : r the number of sets of randomized sobol sequence, n the sample size for each set
# np_MoldIndex a numpy array of mould growth index with size of nxr,
# np_Climate a numpy array of the climate realization for the stochastic cases with size of nxr
def Estimate_Error_Climate (r, n, np_MoldIndex, np_Climate):
    r_Coefficient = 1/(r*(r-1))
    list_Group_Climate = list()
    for i in range (15):
        list_Index_Climate = Group_Climate (np_Climate, i+1, n)
        list_Group_Climate.append(list_Index_Climate)

    np_MouldIndex_Climate_R_mean = np.empty([10])
    np_MouldIndex_Climate_R_std = np.empty([10])

    np_MouldIndex_Climate_E_mean = np.empty([15])
    np_MouldIndex_Climate_E_std = np.empty([15])

    np_MouldIndex_Climate_diff_E_R_mean = np.empty([10])
    np_MouldIndex_Climate_diff_E_R_std = np.empty([10])

    np_MoldIndex_Climate_sum_diff_E_R_mean = np.empty([15])
    np_MoldIndex_Climate_sum_diff_E_R_std = np.empty([15])

    np_stderr_Climate_mean = np.empty([15])
    np_stderr_Climate_std = np.empty([15])

    list_MoldIndex_Climate_R_mean = list()
    list_MoldIndex_Climate_R_std = list()

    list_MouldIndex_Climate_diff_E_R_mean = list()
    list_MouldIndex_Climate_diff_E_R_std = list()
```

```python

    for k in range(15):
        for i in range(r):
            np_MouldIndex_Climate = np.empty([len(list_Group_Climate[k][i])])
            for j in range(len(list_Group_Climate[k][i])):
                np_MouldIndex_Climate[j] = np_MoldIndex [list_Group_Climate[k][i][j],i]
            np_MouldIndex_Climate_R_mean[i] = np.mean(np_MouldIndex_Climate, axis = 0)
            np_MouldIndex_Climate_R_std[i] = np.std(np_MouldIndex_Climate, axis = 0)

        list_MoldIndex_Climate_R_mean.append(np_MouldIndex_Climate_R_mean)
        list_MoldIndex_Climate_R_std.append(np_MouldIndex_Climate_R_std)
        np_MouldIndex_Climate_E_mean[k] = np.mean(np_MouldIndex_Climate_R_mean)
        np_MouldIndex_Climate_E_std[k] = np.mean(np_MouldIndex_Climate_R_std)

        for i in range(r):

            np_MouldIndex_Climate_diff_E_R_mean[i] = math.pow(np_MouldIndex_Climate_R_mean[i] - np_MouldIndex_Climate_E_mean[k],2)
            np_MouldIndex_Climate_diff_E_R_std[i] = math.pow(np_MouldIndex_Climate_R_std[i] - np_MouldIndex_Climate_E_std[k],2)

        list_MouldIndex_Climate_diff_E_R_mean.append(np_MouldIndex_Climate_diff_E_R_mean)
        list_MouldIndex_Climate_diff_E_R_std.append(np_MouldIndex_Climate_diff_E_R_std)

        np_MoldIndex_Climate_sum_diff_E_R_mean[k] = sum (np_MouldIndex_Climate_diff_E_R_mean)
        np_MoldIndex_Climate_sum_diff_E_R_std[k] = sum (np_MouldIndex_Climate_diff_E_R_std)

        np_stderr_Climate_mean[k] = math.pow(r_Coefficient * np_MoldIndex_Climate_sum_diff_E_R_mean[k] , 0.5)
        np_stderr_Climate_std[k] = math.pow(r_Coefficient * np_MoldIndex_Climate_sum_diff_E_R_std[k] , 0.5)


    return np_MouldIndex_Climate_E_mean, np_MouldIndex_Climate_E_std,np_stderr_Climate_mean,np_stderr_Climate_std

```

```python

# This function estimates the standard error of different FD levels
# The inputs are : r the number of sets of randomized sobol sequence, n the sample size for each set
# np_MoldIndex a numpy array of mould growth index with size of nxr,
# np_FD a numpy array of the FD level for the stochastic cases with size of nxr
def Estimate_Error_FD(r, n, np_MoldIndex, np_FD):
    r_Coefficient = 1/(r*(r-1))
    list_Group_FD = list()
    list_FD = ['L','M','H']
    for i in list_FD:
        list_Index_FD = Group_FD (np_FD, i, n)
        list_Group_FD.append(list_Index_FD)

    np_MouldIndex_FD_R_mean = np.empty([10])
    np_MouldIndex_FD_R_std = np.empty([10])

    np_MouldIndex_FD_E_mean = np.empty([3])
    np_MouldIndex_FD_E_std = np.empty([3])

    np_MouldIndex_FD_diff_E_R_mean = np.empty([10])
    np_MouldIndex_FD_diff_E_R_std = np.empty([10])

    np_MoldIndex_FD_sum_diff_E_R_mean = np.empty([3])
    np_MoldIndex_FD_sum_diff_E_R_std = np.empty([3])

    np_stderr_FD_mean = np.empty([3])
    np_stderr_FD_std = np.empty([3])

    list_MoldIndex_FD_R_mean = list()
    list_MoldIndex_FD_R_std = list()
    list_MoldIndex_FD_R_Category=list()
    list_MoldIndex_FD_Category=list()


    list_MouldIndex_FD_diff_E_R_mean = list()
    list_MouldIndex_FD_diff_E_R_std = list()
    list_MoldIndex_FD_Category=list()

```

```
    for k in range(3):
        for i in range(r):
            np_MouldIndex_FD = np.empty([len(list_Group_FD[k][i])])
            for j in range(len(list_Group_FD[k][i])):
                np_MouldIndex_FD[j] = np_MoldIndex [list_Group_FD[k][i][j],i]
            np_MouldIndex_FD_R_mean[i] = np.mean(np_MouldIndex_FD, axis = 0)
            np_MouldIndex_FD_R_std[i] = np.std(np_MouldIndex_FD, axis = 0)
            list_MoldIndex_FD_R_Category. append(np_MouldIndex_FD)

        list_MoldIndex_FD_R_mean.append(np_MouldIndex_FD_R_mean)
        list_MoldIndex_FD_R_std.append(np_MouldIndex_FD_R_std)
        list_MoldIndex_FD_Category. append(list_MoldIndex_FD_R_Category)
        list_MoldIndex_FD_R_Category = list()
        np_MouldIndex_FD_E_mean[k] = np.mean(np_MouldIndex_FD_R_mean)
        np_MouldIndex_FD_E_std[k] = np.mean(np_MouldIndex_FD_R_std)

        for i in range(r):

            np_MouldIndex_FD_diff_E_R_mean[i] = math.pow(np_MouldIndex_FD_R_mean[i] - np_MouldIndex_FD_E_mean[k],2)
            np_MouldIndex_FD_diff_E_R_std[i] = math.pow(np_MouldIndex_FD_R_std[i] - np_MouldIndex_FD_E_std[k],2)

        list_MouldIndex_FD_diff_E_R_mean.append(np_MouldIndex_FD_diff_E_R_mean)
        list_MouldIndex_FD_diff_E_R_std.append(np_MouldIndex_FD_diff_E_R_std)

        np_MoldIndex_FD_sum_diff_E_R_mean[k] = sum (np_MouldIndex_FD_diff_E_R_mean)
        np_MoldIndex_FD_sum_diff_E_R_std[k] = sum (np_MouldIndex_FD_diff_E_R_std)

        np_stderr_FD_mean[k] = math.pow(r_Coefficient * np_MoldIndex_FD_sum_diff_E_R_mean[k] , 0.5)
        np_stderr_FD_std[k] = math.pow(r_Coefficient * np_MoldIndex_FD_sum_diff_E_R_std[k] , 0.5)


    return np_MouldIndex_FD_E_mean, np_MouldIndex_FD_E_std,np_stderr_FD_mean,np_stderr_FD_std,list_MoldIndex_FD_Category
```

```

# This function estimates the standard error of different MS levels
# The inputs are : r the number of sets of randomized sobol sequence, n the sample size for each set
# np_MoldIndex a numpy array of mould growth index with size of nxr,
# np_MS a numpy array of the MS level for the stochastic cases with size of nxr
def Estimate_Error_MS(r, n, np_MoldIndex, np_MS):
    r_Coefficient = 1/(r*(r-1))
    list_Group_MS = list()
    list_MS = ['L','M','H']
    for i in list_MS:
        list_Index_MS = Group_MS (np_MS, i, n)
        list_Group_MS.append(list_Index_MS)

    np_MouldIndex_MS_R_mean = np.empty([10])
    np_MouldIndex_MS_R_std = np.empty([10])

    np_MouldIndex_MS_E_mean = np.empty([3])
    np_MouldIndex_MS_E_std = np.empty([3])

    np_MouldIndex_MS_diff_E_R_mean = np.empty([10])
    np_MouldIndex_MS_diff_E_R_std = np.empty([10])

    np_MoldIndex_MS_sum_diff_E_R_mean = np.empty([3])
    np_MoldIndex_MS_sum_diff_E_R_std = np.empty([3])

    np_stderr_MS_mean = np.empty([3])
    np_stderr_MS_std = np.empty([3])

    list_MoldIndex_MS_R_mean = list()
    list_MoldIndex_MS_R_std = list()
    list_MoldIndex_MS_R_Category=list()
    list_MoldIndex_MS_Category=list()


    list_MouldIndex_MS_diff_E_R_mean = list()
    list_MouldIndex_MS_diff_E_R_std = list()
    list_MoldIndex_MS_Category=list()
```

```python

    for k in range(3):
        for i in range(r):
            np_MouldIndex_MS = np.empty([len(list_Group_MS[k][i])])
            for j in range(len(list_Group_MS[k][i])):
                np_MouldIndex_MS[j] = np_MoldIndex [list_Group_MS[k][i][j],i]
            np_MouldIndex_MS_R_mean[i] = np.mean(np_MouldIndex_MS, axis = 0)
            np_MouldIndex_MS_R_std[i] = np.std(np_MouldIndex_MS, axis = 0)
            list_MoldIndex_MS_R_Category. append(np_MouldIndex_MS)


        list_MoldIndex_MS_R_mean.append(np_MouldIndex_MS_R_mean)
        list_MoldIndex_MS_R_std.append(np_MouldIndex_MS_R_std)
        list_MoldIndex_MS_Category. append(list_MoldIndex_MS_R_Category)
        list_MoldIndex_MS_R_Category = list()
        np_MouldIndex_MS_E_mean[k] = np.mean(np_MouldIndex_MS_R_mean)
        np_MouldIndex_MS_E_std[k] = np.mean(np_MouldIndex_MS_R_std)

        for i in range(r):

            np_MouldIndex_MS_diff_E_R_mean[i] = math.pow(np_MouldIndex_MS_R_mean[i] - np_MouldIndex_MS_E_mean[k],2)
            np_MouldIndex_MS_diff_E_R_std[i] = math.pow(np_MouldIndex_MS_R_std[i] - np_MouldIndex_MS_E_std[k],2)

        list_MouldIndex_MS_diff_E_R_mean.append(np_MouldIndex_MS_diff_E_R_mean)
        list_MouldIndex_MS_diff_E_R_std.append(np_MouldIndex_MS_diff_E_R_std)

        np_MoldIndex_MS_sum_diff_E_R_mean[k] = sum (np_MouldIndex_MS_diff_E_R_mean)
        np_MoldIndex_MS_sum_diff_E_R_std[k] = sum (np_MouldIndex_MS_diff_E_R_std)

        np_stderr_MS_mean[k] = math.pow(r_Coefficient * np_MoldIndex_MS_sum_diff_E_R_mean[k] , 0.5)
        np_stderr_MS_std[k] = math.pow(r_Coefficient * np_MoldIndex_MS_sum_diff_E_R_std[k] , 0.5)


    return np_MouldIndex_MS_E_mean, np_MouldIndex_MS_E_std,np_stderr_MS_mean,np_stderr_MS_std,list_MoldIndex_MS_Category

```

```python
# This function estimates the standard error of different MS levels
# The inputs are : r the number of sets of randomized sobol sequence, n the sample size for each set
# np_MoldIndex a numpy array of mould growth index with size of nxr,
# np_ACH a numpy array of the ACH level for the stochastic cases with size of nxr
def Estimate_Error_ACH(r, n, np_MoldIndex, np_ACH):
    r_Coefficient = 1/(r*(r-1))
    list_Group_ACH = list()
    list_ACH = ['V','2','4']
    for i in list_ACH:
        list_Index_ACH = Group_ACH (np_ACH, i, n)
        list_Group_ACH.append(list_Index_ACH)

    np_MouldIndex_ACH_R_mean = np.empty([10])
    np_MouldIndex_ACH_R_std = np.empty([10])

    np_MouldIndex_ACH_E_mean = np.empty([3])
    np_MouldIndex_ACH_E_std = np.empty([3])

    np_MouldIndex_ACH_diff_E_R_mean = np.empty([10])
    np_MouldIndex_ACH_diff_E_R_std = np.empty([10])

    np_MoldIndex_ACH_sum_diff_E_R_mean = np.empty([3])
    np_MoldIndex_ACH_sum_diff_E_R_std = np.empty([3])

    np_stderr_ACH_mean = np.empty([3])
    np_stderr_ACH_std = np.empty([3])

    list_MoldIndex_ACH_R_mean = list()
    list_MoldIndex_ACH_R_std = list()
    list_MoldIndex_ACH_R_Category=list()
    list_MoldIndex_ACH_Category=list()


    list_MouldIndex_ACH_diff_E_R_mean = list()
    list_MouldIndex_ACH_diff_E_R_std = list()
    list_MoldIndex_ACH_Category=list()

```

```python

    for k in range(3):
        for i in range(r):
            np_MouldIndex_ACH = np.empty([len(list_Group_ACH[k][i])])
            for j in range(len(list_Group_ACH[k][i])):
                np_MouldIndex_ACH[j] = np_MoldIndex [list_Group_ACH[k][i][j],i]
            np_MouldIndex_ACH_R_mean[i] = np.mean(np_MouldIndex_ACH, axis = 0)
            np_MouldIndex_ACH_R_std[i] = np.std(np_MouldIndex_ACH, axis = 0)
            list_MoldIndex_ACH_R_Category. append(np_MouldIndex_ACH)


        list_MoldIndex_ACH_R_mean.append(np_MouldIndex_ACH_R_mean)
        list_MoldIndex_ACH_R_std.append(np_MouldIndex_ACH_R_std)
        list_MoldIndex_ACH_Category. append(list_MoldIndex_ACH_R_Category)
        list_MoldIndex_ACH_R_Category = list()
        np_MouldIndex_ACH_E_mean[k] = np.mean(np_MouldIndex_ACH_R_mean)
        np_MouldIndex_ACH_E_std[k] = np.mean(np_MouldIndex_ACH_R_std)

        for i in range(r):

            np_MouldIndex_ACH_diff_E_R_mean[i] = math.pow(np_MouldIndex_ACH_R_mean[i] - np_MouldIndex_ACH_E_mean[k],2)
            np_MouldIndex_ACH_diff_E_R_std[i] = math.pow(np_MouldIndex_ACH_R_std[i] - np_MouldIndex_ACH_E_std[k],2)

        list_MouldIndex_ACH_diff_E_R_mean.append(np_MouldIndex_ACH_diff_E_R_mean)
        list_MouldIndex_ACH_diff_E_R_std.append(np_MouldIndex_ACH_diff_E_R_std)

        np_MoldIndex_ACH_sum_diff_E_R_mean[k] = sum (np_MouldIndex_ACH_diff_E_R_mean)
        np_MoldIndex_ACH_sum_diff_E_R_std[k] = sum (np_MouldIndex_ACH_diff_E_R_std)

        np_stderr_ACH_mean[k] = math.pow(r_Coefficient * np_MoldIndex_ACH_sum_diff_E_R_mean[k] , 0.5)
        np_stderr_ACH_std[k] = math.pow(r_Coefficient * np_MoldIndex_ACH_sum_diff_E_R_std[k] , 0.5)


    return np_MouldIndex_ACH_E_mean, np_MouldIndex_ACH_E_std,np_stderr_ACH_mean,np_stderr_ACH_std,list_MoldIndex_ACH_Category

```

```python
# This function group the stochastic cases based on orientation
# The inputs are : np_Orientation- a numpy array of the orientations of the stochastic cases with size of nxr,
# Orientation- the specific orientation to be grouped
def Group_Orientation (np_Orientation, Orientation, n):
    list_InputIndex_Orientation = list()
    list_InputIndex_Orientation_R = list()

    for i in range(10):
        for j in range(n):
            if np_Orientation[j,i] == Orientation:
               list_InputIndex_Orientation_R.append(j)
        list_InputIndex_Orientation.append(list_InputIndex_Orientation_R)
        list_InputIndex_Orientation_R = list()

    return list_InputIndex_Orientation

# This function group the stochastic cases based on climate realization
# The inputs are : np_Climate- a numpy array of the climate realizations of the stochastic cases with size of nxr,
# Climate_Realization- the specific orientation to be grouped
def Group_Climate (np_Climate, Climatic_Realization, n):
    list_InputIndex_Climate = list()
    list_InputIndex_Climate_R = list()

    for i in range(10):
        for j in range(n):
            if np_Climate[j,i] == Climatic_Realization:
               list_InputIndex_Climate_R.append(j)
        list_InputIndex_Climate.append(list_InputIndex_Climate_R)
        list_InputIndex_Climate_R = list()

    return list_InputIndex_Climate
```

```python
# This function group the stochastic cases based on FD
# The inputs are : np_FD_Categories- a numpy array of the FD levels of the stochastic cases with size of nxr,
# FD_Category- the level of FD to be grouped
def Group_FD (np_FD_Categories, FD_Category, n):
    list_InputIndex_FD = list()
    list_InputIndex_FD_R = list()

    for i in range(10):
        for j in range(n):
            if np_FD_Categories [j,i] == FD_Category:
                list_InputIndex_FD_R.append(j)
        list_InputIndex_FD.append(list_InputIndex_FD_R)
        list_InputIndex_FD_R = list()
    return list_InputIndex_FD

# This function group the stochastic cases based on MS
# The inputs are : np_MS_Categories- a numpy array of the MS levels of the stochastic cases with size of nxr,
#  MS_Category- the level of MS to be grouped
def Group_MS (np_MS_Categories, MS_Category, n):
    list_InputIndex_MS = list()
    list_InputIndex_MS_R = list()

    for i in range(10):
        for j in range(n):
            if np_MS_Categories [j,i] == MS_Category:
                list_InputIndex_MS_R.append(j)
        list_InputIndex_MS.append(list_InputIndex_MS_R)
        list_InputIndex_MS_R = list()
    return list_InputIndex_MS

```

```python
# This function group the stochastic cases based on ACH
# The inputs are : np_ACH_Categories- a numpy array of the ACH levels of the stochastic cases with size of nxr,
# ACH_Category- the level of ACH to be grouped
def Group_ACH (np_ACH_Categories, ACH_Category, n):
    list_InputIndex_ACH = list()
    list_InputIndex_ACH_R = list()

    for i in range(10):
        for j in range(n):
            if np_ACH_Categories [j,i] == ACH_Category:
                list_InputIndex_ACH_R.append(j)
        list_InputIndex_ACH.append(list_InputIndex_ACH_R)
        list_InputIndex_ACH_R = list()
    return list_InputIndex_ACH
```

2. Main routine for error estimation

2.1 Error estimation for different orientation and climate realizations

```python
import pandas as pd
from Error_Estimation import Estimate_Error
from Error_Estimation import Estimate_Error_Orientation
from Error_Estimation import Estimate_Error_Climate
from Error_Estimation import Estimate_Error_FD
from Error_Estimation import Estimate_Error_MS
from Error_Estimation import Estimate_Error_ACH
import numpy as np

##################################### Error estimation for different orientations and climate realizations ##########################
# Load stochastic results
pd_F7 = pd.read_csv('D:/StochasticModelling/StochasticScripts/Stochastic_Outputs/MoldIndex_Ave_2560.csv')
pd_F7.set_index('Unnamed: 0', inplace = True)
np_F7 = pd_F7.to_numpy()

# Estimate the standard error at different sample size for the whole sample space
stderr_F7_80 = Estimate_Error(10,8,np_F7)
stderr_F7_160 = Estimate_Error(10,16,np_F7)
stderr_F7_320 = Estimate_Error(10,32,np_F7)
stderr_F7_640 = Estimate_Error(10,64,np_F7)
stderr_F7_1280 = Estimate_Error(10,128,np_F7)
stderr_F7_1920 = Estimate_Error(10,192,np_F7)
stderr_F7_2560 = Estimate_Error(10,256,np_F7)

# Load the orientations
pd_Orientation = pd.read_csv('D:/StochasticModelling/StochasticScripts/Stochastic_Inputs/Stochastic_Orientation_256.csv')
np_Orientation = pd_Orientation.to_numpy()
```

```python
# Estimate the standard error at different sample size for different orientations
stderr_F7_Group_Orientation_1600 = Estimate_Error_Orientation(10,160,np_F7,np_Orientation )
stderr_F7_Group_Orientation_1920 = Estimate_Error_Orientation(10,192,np_F7,np_Orientation )
stderr_F7_Group_Orientation_2560 = Estimate_Error_Orientation(10,256,np_F7,np_Orientation )

# Load the climate realizations
list_Stochastic_Variables = list()
np_Climatic_Realizations = np.empty([256,10],dtype = int)
for i in range(10):
    pd_Stochastic_Variables = pd.read_csv\
        ('D:/StochasticModelling/StochasticScripts/Stochastic_Inputs/Stochastic_Inputs_256/Stochastic_Variables_Simulation_256_'+
         str(i+1)+'.csv')
    np_Stochastic_Variables = pd_Stochastic_Variables.to_numpy()
    list_Stochastic_Variables.append(np_Stochastic_Variables)

for i in range(10):
    for j in range(256):
        np_Climatic_Realizations [j,i] = list_Stochastic_Variables[i][j,0]
```

2.2 Error estimation for different levels of rain deposition factor, moisture source and cladding ventilation

```python
######################### Error estimation for fixed orientation, different levels of FD, MS and ACH ################################

#Load stochastic results
pd_F7 = pd.read_csv('D:/StochasticModelling/StochasticScripts/Stochastic_Outputs/Fix_Orientation/MoldIndex_Ave_1280.csv')
pd_F7.set_index('Unnamed: 0', inplace = True)
np_F7 = pd_F7.to_numpy()

#Load stochastic inputs
list_Stochastic_Variables = list()
for i in range(10):
    pd_Stochastic_Variables = pd.read_csv\
        ('D:/StochasticModelling/StochasticScripts/Stochastic_Inputs/Stochastic_Inputs_FixOrientation/Stochastic_Variables_PostProcess
         +str(i+1)+'.csv')
    np_Stochastic_Variables = pd_Stochastic_Variables.to_numpy()
    list_Stochastic_Variables.append(np_Stochastic_Variables)

# Error estimation at different FD
np_FD = np.empty([192,10], dtype = str)

for i in range(10):
    for j in range(192):
        np_FD[j,i] = list_Stochastic_Variables[i][j,9]

list_Group_FD_640 = Estimate_Error_FD (10, 64, np_F7, np_FD)
list_Group_FD_320 = Estimate_Error_FD (10, 32, np_F7, np_FD)
list_Group_FD_160 = Estimate_Error_FD (10, 16, np_F7, np_FD)
list_Group_FD_80 = Estimate_Error_FD (10, 8, np_F7, np_FD)

np_FD_Low = list_Group_FD_640[4][0][0]
np_FD_Middle = list_Group_FD_640[4][1][0]
np_FD_High = list_Group_FD_640[4][2][0]

for i in range (1,10):
    np_FD_Low = np.concatenate((np_FD_Low,list_Group_FD_160[4][0][i]))
    np_FD_Middle = np.concatenate((np_FD_Middle,list_Group_FD_640[4][1][i]))
    np_FD_High = np.concatenate((np_FD_High,list_Group_FD_640[4][2][i]))
```

```
# Error estimation at different MS
np_MS = np.empty([192,10], dtype = str)

for i in range(10):
    for j in range(192):
        np_MS[j,i] = list_Stochastic_Variables[i][j,8]

list_Group_MS_640 = Estimate_Error_MS (10, 64, np_F7, np_MS)
list_Group_MS_320 = Estimate_Error_MS (10, 32, np_F7, np_MS)
list_Group_MS_160 = Estimate_Error_MS (10, 16, np_F7, np_MS)
list_Group_MS_80 = Estimate_Error_MS (10, 8, np_F7, np_MS)

np_MS_Low = list_Group_MS_640[4][0][0]
np_MS_Middle = list_Group_MS_640[4][1][0]
np_MS_High = list_Group_MS_640[4][2][0]

for i in range (1,10):
    np_MS_Low = np.concatenate((np_MS_Low,list_Group_MS_640[4][0][i]))
    np_MS_Middle = np.concatenate((np_MS_Middle,list_Group_MS_640[4][1][i]))
    np_MS_High = np.concatenate((np_MS_High,list_Group_MS_640[4][2][i]))

# Error estimation at different ACH
np_ACH = np.empty([192,10], dtype = str)

for i in range(10):
    for j in range(192):
        np_ACH[j,i] = list_Stochastic_Variables[i][j,10]

list_Group_ACH_640 = Estimate_Error_ACH (10, 64, np_F7, np_ACH)
list_Group_ACH_320 = Estimate_Error_ACH (10, 32, np_F7, np_ACH)
list_Group_ACH_160 = Estimate_Error_ACH (10, 16, np_F7, np_ACH)
list_Group_ACH_80 = Estimate_Error_ACH (10, 8, np_F7, np_ACH)

np_ACH_Low = list_Group_ACH_640[4][0][0]
np_ACH_Middle = list_Group_ACH_640[4][1][0]
np_ACH_High = list_Group_ACH_640[4][2][0]

for i in range (1,10):
    np_ACH_Low = np.concatenate((np_ACH_Low,list_Group_ACH_640[4][0][i]))
    np_ACH_Middle = np.concatenate((np_ACH_Middle,list_Group_ACH_640[4][1][i]))
    np_ACH_High = np.concatenate((np_ACH_High,list_Group_ACH_640[4][2][i]))
```

Appendix I Scripts for visualizing the stochastic results

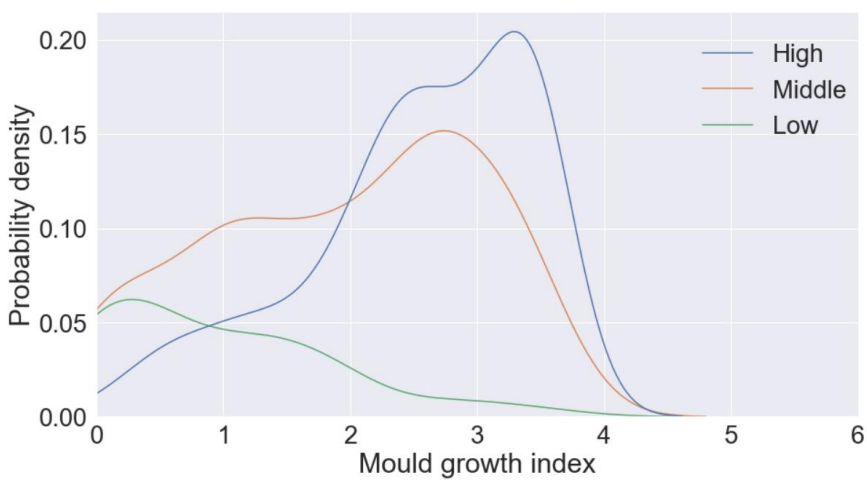
```python
import pandas as pd
import numpy as np
import seaborn as sns
from Error_Estimation import Estimate_Error_FD
from Error_Estimation import Estimate_Error_MS
from Error_Estimation import Estimate_Error_ACH
import matplotlib.pyplot as plt

# Load stochastic results
pd_F7 = pd.read_csv('D:/StochasticModelling/StochasticScripts/Stochastic_Outputs/Fix_Orientation/MoldIndex_Ave_1280.csv')
pd_F7.set_index('Unnamed: 0', inplace = True)
np_F7 = pd_F7.to_numpy()

# Load stochastic inputs
list_Stochastic_Variables = list()
for i in range(10):
    pd_Stochastic_Variables = pd.read_csv('D:/StochasticModelling/Building_Special_Issue/Stochastic_Inputs/Stochastic_Inputs_FixOrient
    np_Stochastic_Variables = pd_Stochastic_Variables.to_numpy()
    list_Stochastic_Variables.append(np_Stochastic_Variables)

# Seperate the results for different levels of FD
np_FD = np.empty([192,10], dtype = str)

for i in range(10):
    for j in range(192):
        np_FD[j,i] = list_Stochastic_Variables[i][j,9]

list_Group_FD_640 = Estimate_Error_FD (10, 64, np_F7, np_FD)
list_Group_FD_320 = Estimate_Error_FD (10, 32, np_F7, np_FD)
list_Group_FD_160 = Estimate_Error_FD (10, 16, np_F7, np_FD)
list_Group_FD_80 = Estimate_Error_FD (10, 8, np_F7, np_FD)

np_FD_Low = list_Group_FD_640[4][0][0]
np_FD_Middle = list_Group_FD_640[4][1][0]
np_FD_High = list_Group_FD_640[4][2][0]

for i in range (1,10):
    np_FD_Low = np.concatenate((np_FD_Low,list_Group_FD_160[4][0][i]))
    np_FD_Middle = np.concatenate((np_FD_Middle,list_Group_FD_640[4][1][i]))
    np_FD_High = np.concatenate((np_FD_High,list_Group_FD_640[4][2][i]))
```

```python
# Seperate the results for different levels of MS
np_MS = np.empty([192,10], dtype = str)
for i in range(10):
    for j in range(192):
        np_MS[j,i] = list_Stochastic_Variables[i][j,8]

list_Group_MS_640 = Estimate_Error_MS (10, 64, np_F7, np_MS)
list_Group_MS_320 = Estimate_Error_MS (10, 32, np_F7, np_MS)
list_Group_MS_160 = Estimate_Error_MS (10, 16, np_F7, np_MS)
list_Group_MS_80 = Estimate_Error_MS (10, 8, np_F7, np_MS)

np_MS_Low = list_Group_MS_640[4][0][0]
np_MS_Middle = list_Group_MS_640[4][1][0]
np_MS_High = list_Group_MS_640[4][2][0]

for i in range (1,10):
    np_MS_Low = np.concatenate((np_MS_Low,list_Group_MS_640[4][0][i]))
    np_MS_Middle = np.concatenate((np_MS_Middle,list_Group_MS_640[4][1][i]))
    np_MS_High = np.concatenate((np_MS_High,list_Group_MS_640[4][2][i]))

# Seperate the results for different levels of ACH
np_ACH = np.empty([192,10], dtype = str)
for i in range(10):
    for j in range(192):
        np_ACH[j,i] = list_Stochastic_Variables[i][j,10]

list_Group_ACH_640 = Estimate_Error_ACH (10, 64, np_F7, np_ACH)
list_Group_ACH_320 = Estimate_Error_ACH (10, 32, np_F7, np_ACH)
list_Group_ACH_160 = Estimate_Error_ACH (10, 16, np_F7, np_ACH)
list_Group_ACH_80 = Estimate_Error_ACH (10, 8, np_F7, np_ACH)

np_ACH_Low = list_Group_ACH_640[4][0][0]
np_ACH_Middle = list_Group_ACH_640[4][1][0]
np_ACH_High = list_Group_ACH_640[4][2][0]

for i in range (1,10):
    np_ACH_Low = np.concatenate((np_ACH_Low,list_Group_ACH_640[4][0][i]))
    np_ACH_Middle = np.concatenate((np_ACH_Middle,list_Group_ACH_640[4][1][i]))
    np_ACH_High = np.concatenate((np_ACH_High,list_Group_ACH_640[4][2][i]))
```

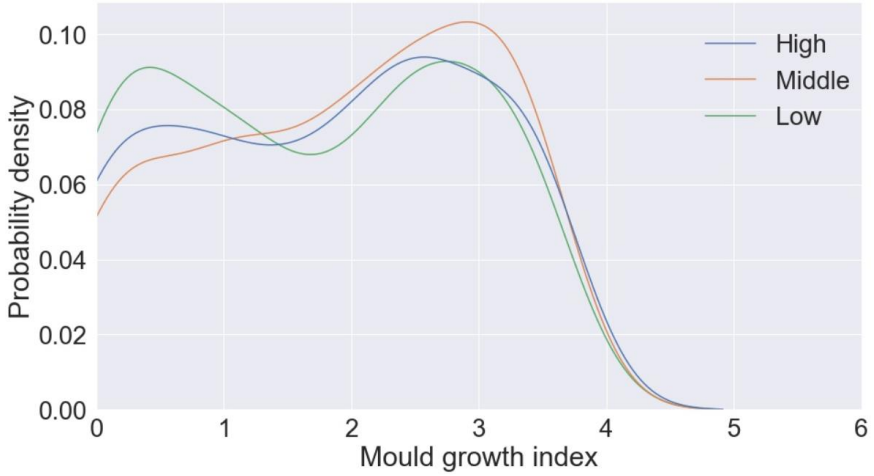
```python
# Box plot #
list_High = ['High']
list_Middle = ['Middle']
list_Low = ['Low']
df_MS_High = pd.DataFrame(np_MS_High, columns = list_High ).assign(Trial = 'MS').assign(Level = 'High')
df_MS_Middle = pd.DataFrame(np_MS_Middle, columns = list_Middle).assign(Trial = 'MS').assign(Level = 'Middle')
df_MS_Low = pd.DataFrame(np_MS_Low, columns = list_Low).assign(Trial = 'MS').assign(level ='Low')

df_FD_High = pd.DataFrame(np_FD_High,columns = list_High).assign(Trial = 'FD').assign(Level = 'High')
df_FD_Middle = pd.DataFrame(np_FD_Middle, columns = list_Middle).assign(Trial = 'FD').assign(Level = 'Middle')
df_FD_Low = pd.DataFrame(np_FD_Low, columns = list_Low).assign(Trial = 'FD').assign(level ='Low')

df_ACH_High = pd.DataFrame(np_ACH_High,columns = list_High).assign(Trial = 'ACH').assign(Level = 'High')
df_ACH_Middle = pd.DataFrame(np_ACH_Middle, columns = list_Middle).assign(Trial = 'ACH').assign(Level = 'Middle')
df_ACH_Low = pd.DataFrame(np_ACH_Low, columns = list_Low).assign(Trial = 'ACH').assign(level ='Low')

np_FD_High_new = df_FD_High.to_numpy()
np_FD_Middle_new = df_FD_Middle.to_numpy()
np_FD_Low_new = df_FD_Low.to_numpy()

np_MS_High_new = df_MS_High.to_numpy()
np_MS_Middle_new = df_MS_Middle.to_numpy()
np_MS_Low_new = df_MS_Low.to_numpy()

np_ACH_High_new = df_ACH_High.to_numpy()
np_ACH_Middle_new = df_ACH_Middle.to_numpy()
np_ACH_Low_new = df_ACH_Low.to_numpy()

list_Boxes =  ['Mould growth index','Variables','Level']
np_BoxPlot = np.concatenate((np_FD_High_new ,np_FD_Middle_new,np_FD_Low_new,np_MS_High_new,np_MS_Middle_new ,
                             np_MS_Low_new,np_ACH_High_new,np_ACH_Middle_new ,np_ACH_Low_new ))
pd_BoxPlot = pd.DataFrame(data = np_BoxPlot, columns = list_Boxes)

plt.figure(figsize=(14, 8))
ax = sns.boxplot(x = 'Variables', y = 'Mould growth index', hue = 'Level', data = pd_BoxPlot)
ax.set_xlabel(" ",fontsize=25)
ax.set_ylabel("Mould growth index",fontsize=25)
ax.tick_params(labelsize=25)
plt.show
```

```python
plt.legend(loc='upper right')
plt.setp(ax.get_legend().get_texts(), fontsize='25')
plt.setp(ax.get_legend().get_title(), fontsize='25')
plt.ylim(0, 6)

## PDF and CDF #
# Categorize results based on different design parameters
np_FD_Categorized = np.concatenate((np_FD_High_new ,np_FD_Middle_new,np_FD_Low_new))
pd_FD_Categorized = pd.DataFrame(data = np_FD_Categorized , columns = list_Boxes)
pd_FD_Categorized["Mould growth index"] = pd.to_numeric(pd_FD_Categorized["Mould growth index"], downcast="float")

np_MS_Categorized = np.concatenate((np_MS_High_new ,np_MS_Middle_new,np_MS_Low_new))
pd_MS_Categorized = pd.DataFrame(data = np_MS_Categorized , columns = list_Boxes)
pd_MS_Categorized["Mould growth index"] = pd.to_numeric(pd_MS_Categorized["Mould growth index"], downcast="float")

np_ACH_Categorized = np.concatenate((np_ACH_High_new ,np_ACH_Middle_new,np_ACH_Low_new))
pd_ACH_Categorized = pd.DataFrame(data = np_ACH_Categorized , columns = list_Boxes)
pd_ACH_Categorized["Mould growth index"] = pd.to_numeric(pd_ACH_Categorized["Mould growth index"], downcast="float")

# Probability density plot #
sns.set(font_scale = 2.5)
ax=sns.displot(data=pd_FD_Categorized, x='Mould growth index', hue = 'Level', kind='kde',color = 'darkblue',legend =True,
               height = 8, aspect = 14/8)
ax.set_ylabels ('Probability density')
ax.set(xlim=(0, 6))
leg = ax._legend
leg.set_bbox_to_anchor([0.8, 0.75])  # coordinates of the position of legend
leg.set_title(None)
ax.savefig('D:/StochasticModelling/StochasticScripts/Plots/pdf_FD.jpg')

sns.set(font_scale = 2.5)
ax=sns.displot(data=pd_MS_Categorized, x='Mould growth index', hue = 'Level', kind='kde',color = 'darkblue',legend =True,
               height = 8, aspect = 14/8)
ax.set_ylabels ('Probability density')
ax.set(xlim=(0, 6))
leg = ax._legend
leg.set_bbox_to_anchor([0.8, 0.75])
leg.set_title(None)
ax.savefig('D:/StochasticModelling/StochasticScripts/Plots/pdf_MS.jpg')
```

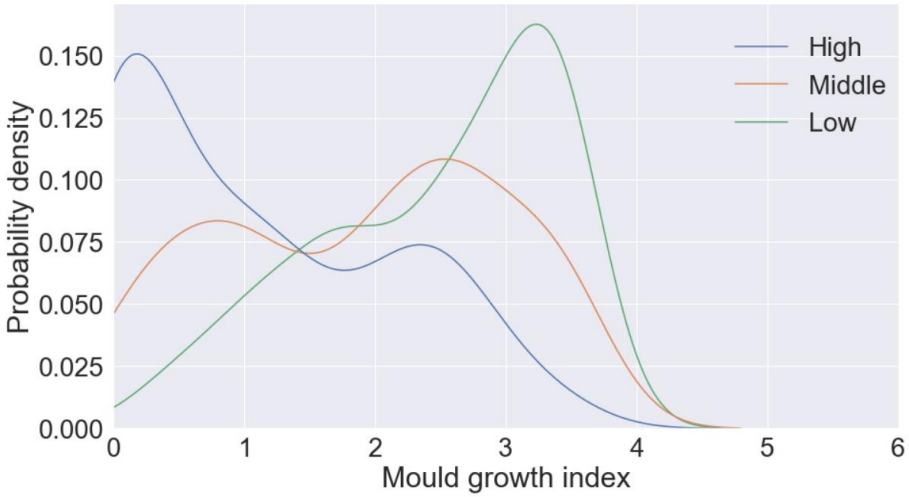
```
sns.set(font_scale = 2.5)
ax=sns.displot(data=pd_ACH_Categorized, x='Mould growth index', hue = 'Level', kind='kde',color = 'darkblue',legend =True,
               height = 8, aspect = 14/8)
ax.set_ylabels ('Probability density')
ax.set(xlim=(0, 6))
leg = ax._legend
leg.set_bbox_to_anchor([0.8, 0.75])
leg.set_title(None)
ax.savefig('D:/StochasticModelling/StochasticScripts/Plots/pdf_ACH.jpg')

# Cumulative distribution plot
sns.set(font_scale = 2.5)
ax=sns.displot(data=pd_FD_Categorized, x='Mould growth index', hue = 'Level', kind='ecdf',color = 'darkblue',legend =True,
               height = 8, aspect = 14/8)
ax.set_ylabels ('Probability density')
ax.set(xlim=(0, 6))
leg = ax._legend
leg.set_bbox_to_anchor([0.8, 0.75])
leg.set_title(None)
ax.savefig('D:/StochasticModelling/StochasticScripts/Plots/cdf_FD.jpg')

sns.set(font_scale = 2.5)
ax=sns.displot(data=pd_MS_Categorized, x='Mould growth index', hue = 'Level', kind='ecdf',color = 'darkblue',legend =True,
               height = 8, aspect = 14/8)
ax.set_ylabels ('Probability density')
ax.set(xlim=(0, 6))
leg = ax._legend
leg.set_bbox_to_anchor([0.8, 0.75])
leg.set_title(None)
ax.savefig('D:/StochasticModelling/StochasticScripts/Plots/cdf_MS.jpg')

sns.set(font_scale = 2.5)
ax=sns.displot(data=pd_ACH_Categorized, x='Mould growth index', hue = 'Level', kind='ecdf',color = 'darkblue',legend =True,
               height = 8, aspect = 14/8)
ax.set_ylabels ('Probability density')
ax.set(xlim=(0, 6))
leg = ax._legend
leg.set_bbox_to_anchor([0.8, 0.75])
leg.set_title(None)
ax.savefig('D:/StochasticModelling/StochasticScripts/Plots/cdf_ACH.jpg')
```



a) Rain deposition factor

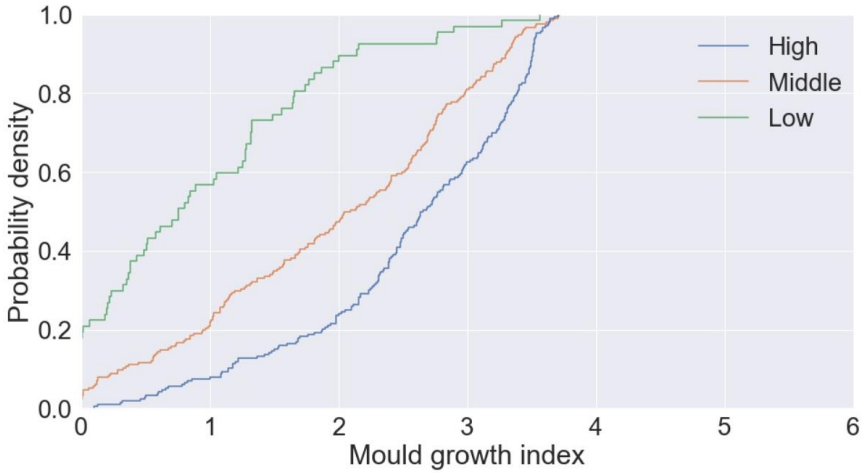


b) Rain leakage moisture source

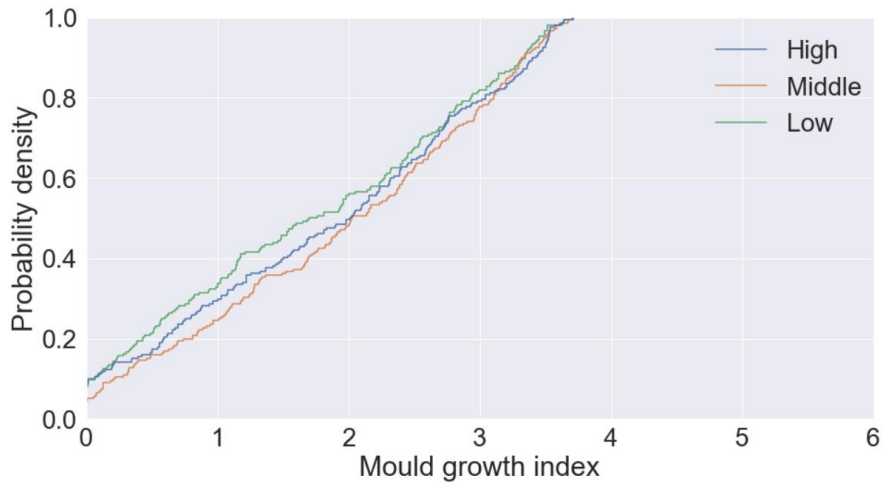


c) Cladding ventilation rate (ACH)

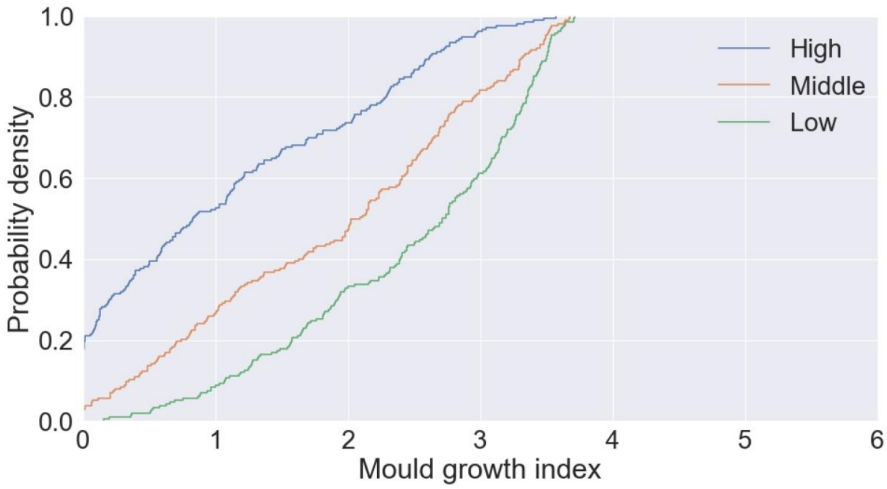
Figure 16 PDF of 2-years' averaged mould growth index



a) Rain deposition factor



b) Rain leakage moisture source



c) Cladding ventilation rate (ACH)

Figure 17 CDF of 2-years' averaged mould growth index