Proceedings of the 35th European Safety and Reliability & the 33rd Society for Risk Analysis Europe Conference
Edited by Eirik Bjorheim Abrahamsen, Terje Aven, Frederic Bouder, Roger Flage, Marja Ylönen
 Published by Research Publishing, Singapore.
doi: 10.3850/978-981-94-3281-3_ESREL-SRA-E2025-P3042-cd

UncertaintyQuantification.jl: Efficient reliability analysis powered by Julia

Jasper Behrensdorf
Institute for Risk and Reliability, Leibniz University Hannover, Germany
E-mail: behrensdorf@irz.uni-hannover.de

Ander Gray
Heudiasyc, Université de technologie de Compiègne, France E-mail: ander.gray@hds.utc.fr

Matteo Broggi
Institute for Risk and Reliability, Leibniz University Hannover, Germany

Michael Beer
Institute for Risk and Reliability, Leibniz University Hannover, Germany
Institute for Risk and Uncertainty, University of Liverpool, Liverpool, UK
International Joint Research Center for Engineering Reliability and Stochastic Mechanics, Tongji University, Shanghai, China

This work presents the latest features and developments in UncertaintyQuantification.jl, a simulation-based and open-source framework for uncertainty quantification and risk analysis written in Julia. The framework has undergone extensive development since its initial release in August 2020, and now includes a number of numerical algorithms for reliability analysis, sensitivity analysis and surrogate modelling. While this paper presents all currently available features, the main emphasis is on the recent introduction of imprecise probabilities. The evaluation of probability boxes and intervals in any numerical model is a feature we believe sets UncertaintyQuantification.jl apart from other software in the field. Another important element is the ability to interface with HPC job schedulers such as Slurm. This makes the software accessible to simulation codes that require multiple compute nodes, and allows for scalable parallelisation in both the uncertainty algorithm and the simulator. Adequate illustrative numerical examples are presented throughout the paper to highlight the capabilities of the implemented algorithms.

Keywords: Uncertainty quantification, Reliability analysis, Imprecise probabilities, Software

1. Introduction

The need for accurate and efficient treatment of uncertainties across many engineering and scientific disciplines has given rise to a number of computational frameworks implemented in a variety of programming languages. *OpenTURNS* (C++/Python) (Baudin et al., 2017), *OpenCossan* (MATLAB) (Patelli et al., 2014), *UQLab* (MATLAB) (Marelli and Sudret, 2014), *UQpy* (Python) (Olivier et al., 2020), and *EasyVVUQ* (Python) (Wright et al., 2020) amongst others.

In this paper we present the latest additions to *UncertaintyQuantification.jl* (Behrensdorf et al., 2024), an alternative framework that has been in development since 2019. This new software is inspired by and extends some capabilities of *OpenCossan*. We build upon the knowledge we have gained throughout the development of *OpenCossan*, while at the same time leveraging the capabilities of the modern, open source, and highly performant Julia programming language (Bezanson et al., 2017).

From its inception until today we have constantly expanded the software with new algorithms and made some 20 releases. What follows is a non-exhaustive list of features currently available:

- Simulation-based reliability analysis[a]
- Local and global sensitivity analysis

[a]Also known as rare-event simulation or variance reduction.

- Third-party solver interface
- High performance computing (Slurm)
- Surrogate modelling
- Bayesian model updating
- Stochastic process modelling
- Imprecise probabilities

This last feature, specifically the propagation of intervals and probability boxes (the numerical treatment of epistemic uncertainty), is what we believe sets us apart from other software in the field, and therefore is the focus of this work. The remainder of this paper is structured as follows. Section 2 introduces the basic notation for imprecise probabilities and how to define them in the framework, after which Section 2 discusses the imprecise reliability analysis. Sections 4 and 5 present two family of algorithms implemented for reliability analysis with imprecise probabilities. A short introduction into the high performance computing feature is provided in Section 7. Section 6 presents a full numerical example, followed by a conclusion.

2. Imprecise Probabilities

Uncertainty is typically classified into two types: *aleatory* and *epistemic* uncertainty. Aleatory uncertainty, also called irreducible uncertainty, describes the inherent randomness of a process. Epistemic uncertainty is the uncertainty resulting from a lack of knowledge, and thus is sometimes called reducible uncertainty as it can be reduced through the acquisition of additional data and information. If both types of uncertainties occur together in the same variable this is sometimes called *hybrid* uncertainty. In *UncertaintyQuantification.jl*, purely aleatory uncertainties are represented using the `RandomVariable` type. To represent purely epistemic uncertainty we provide the `Interval` type.

Hybrid uncertainties can be expressed as probability boxes (p-box), which are interval bounded cumulative distribution functions (CDFs). Consider two CDFs $\underline{F}$ and $\overline{F}$ with $\underline{F}(x) \leq \overline{F}(x)$ for all $x \in \mathbb{R}$. Then, $[\underline{F}(x), \overline{F}(x)]$ is a bound on F such that $\underline{F}(x) \leq F(x) \leq \overline{F}(x)$ is the set of CDFs. This set is called the p-box for an imprecisely known random variable X, where $\underline{F}(x)$ is the lower bound for the probability that X is smaller than or equal to x, and $\overline{F}(x)$ is the upper bound of this probability. The simplest way to construct p-boxes is by passing an `Interval` to the `ProbabilityBox` constructor. See Code 1 for an example definition of a Gaussian p-box with imprecise mean and precise standard deviation. These are known as parametric p-boxes, as some methods can make use of their distribution shape information. If only the outer CDFs are known (perhaps obtained by non-parametric means), then every possible CDF between the bounds (irrespective of family) is included in the set, which can be called a distribution-free p-box (Ferson et al., 2015). Figure 1 presents some examples of typical precise and imprecise inputs.

3. Imprecise Reliability Analysis

In engineering, the term *reliability* is used to describe the ability of a system to perform the function it was designed for under uncertain or varying conditions. The system's state is typically identified using a *performance function* $g(x)$, with failures being defined as non-positive values $g(x) \leq 0$. The *probability of failure* is then defined as the likelihood of the system being in the failed state. In the presence of purely *aleatory* uncertainty this is given as

$$p_f = \int \mathbb{I}[g(x)] f_{\mathbf{X}}(x) dx, \tag{1}$$

where $f_{\mathbf{X}}(x)$ denotes the joint probability density function of the input $\mathbf{X}$ and $\mathbb{I}$ is an indicator function defined as

$$\mathbb{I}[g(x)] = \begin{cases} 0 & \text{if} \quad g(x) > 0 \\ 1 & \text{if} \quad g(x) \leq 0. \end{cases} \tag{2}$$

However, if *epistemic* uncertainty is considered in the input variables it must also be propagated through the analysis, converting the probability of failure into an interval itself. The goal of the reliability analysis is then to find the lower bound $\underline{p}_f$ and upper bound $\overline{p}_f$ of the probability of failure such that $p_f \in [\underline{p}_f, \overline{p}_f]$. Formally, this can be defined as

$$\underline{p}_f = \min_{\vartheta \in \theta} \int \mathbb{I}[g(x)] f_{\mathbf{X}}(x, \vartheta) dx \tag{3}$$

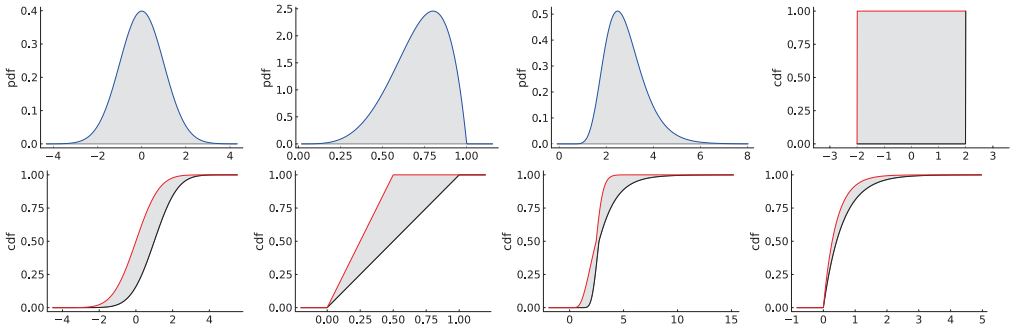

Fig. 1. Examples of precise and imprecise inputs. Sequentially: a standard normal $N(0,1)$, a beta distribution $B(5,2)$, a log-normal $(5,\,2)$, an interval $[-2,2]$, normal p-box with interval mean $N([0,1],1)$, a uniform p-box with an interval upper bound $U(0,[0.5,1])$, a log-normal with interval parameters $\log(N([0.9,1],[0.2,0.5]))$, and an exponential distribution with an interval rate $\theta \in [0.4,0.6]$.

```
x = ProbabilityBox{Normal}([Interval(-1, 2, :μ),Parameter(1, :σ)], :x)
```

Code 1: Definition of a Gaussian p-box.

and

$$\overline{p}_f = \max_{\vartheta \in \theta} \int \mathbb{I}[g(x)] f_{\mathbf{X}}(x,\vartheta) dx, \qquad (4)$$

where θ represents the epistemic parameters characterizing the inputs.

The following two sections provide two possible solutions to this challenging problem.

4. Double-loop Monte Carlo Simulation

The simplest way to solve Equations (3) and (4) is by double-loop simulation. The name refers to the need for two loops in comparison to a standard reliability analysis. The outer-loop essentially solves two optimisation problems over the parameter space of the epistemic inputs to find the combinations that minimize and maximize the probability of failure. The inner-loop requires a reliability calculation, with the current combination of parameters under consideration being mapped to precise inputs. In practice, `Interval` is mapped to a `Parameter` while a `ProbabilityBox` yields a `RandomVariable`. Therefore, the double-loop simulation treats p-boxes as parametric. Then, a comprehensive reliability analysis using these purely *aleatory* inputs is carried out to solve Eq. (1). This repeated analysis in the inner-loop makes the double-loop simulation computationally demanding. If a Monte Carlo simulation is applied in the inner-loop this is known as double-loop Monte Carlo simulation.

Special attention must be paid to the type of optimisation algorithm used, as random sampling in the inner-loop leads to non-smooth objective functions. Here we have chosen to apply mesh adaptive direct search (MADS) algorithms which are specifically designed for such cases (Abramson et al., 2009). However, exploration of alternative methods is part of the ongoing development.

Estimating the probability of failure is effectively separated into two independent problems, one for each bound. This provides the ability to apply different types of analyses. For example, using a larger number of samples for the lower bound.

As an example, consider the function

$$f(x_1,x_2) = x_1 + x_2, \qquad (5)$$

where $x_1 \in [-1,1]$ is an interval and $x_2 \sim N(0,[1,2])$ is distributed as a Gaussian p-box. The associated performance function is

$$g(x) = 9 + f(x),$$

i.e., failures are $f(x) \leq -9$. The analytical solution

to this problem is known to be

$$\underline{p}_f = \Phi(-9; 1, 1)$$

and

$$\overline{p}_f = \Phi(-9; -1, 2),$$

where $\Phi(x; \mu, \sigma)$ is the Gaussian CDF with mean μ and standard deviation σ. Refer to Code 2 for how to set up this problem in *UncertaintyQuantification.jl*.

Reliability analysis is exposed through the `probability_of_failure` function which takes four input arguments. The `Model`, the performance function, the input variables and finally the algorithm to use. To apply a double-loop we pass the inner simulation to be used to the `DoubleLoop`. For this simple example we apply the first order reliability method (FORM), as this can reliably estimate the small failure probability of the lower bound which is $\approx 7.6e-24$. See Code 3 for the complete function call. The bounds on the p_f can be obtained from the output interval through `pf.lb` and `pf.ub`. The epistemic uncertainty is propagated correctly and matches the analytical solution. On top of the probability of failure, the double-loop analysis also returns the parameters that lead to the lower and upper bound. Here, they are correctly identified as $[1, 1]$ and $[-1, 2]$.

5. Random Slicing

An alternative method for computing probability bounds for reliability problems is based on random-set theory, as outlined by Alvarez et al. (2018). In this paper (and in software) we colloquially call this "random slicing", as will become apparent. As opposed to double-loop Monte Carlo, random slicing is a *distribution-free* or a *non-parametric* technique, as it does not make use of distribution parameters or family. For this reason, it is slightly more general, but can provide wider probability intervals in certain simulations.

In random slicing, we make use of the fact that a p-box is a random-set (Ferson et al., 2015) to simulate random realisations (random intervals) from the inverses of the bounding CDFs

$$\Gamma(\alpha) = [\overline{F}^{-1}(\alpha), \underline{F}^{-1}(\alpha)],$$

where $\alpha \sim U(0, 1)$ is a sample from a uniform distribution. This interval can be visualised as a horizontal cut (or slice) of the p-box at an $\alpha \in [0, 1]$. This interval is then propagated through the model f or performance function g using an optimiser or surrogate model,

$$\underline{g}(\alpha) = \min_{x \in \Gamma(\alpha)} g(x),$$

$$\overline{g}(\alpha) = \max_{x \in \Gamma(\alpha)} g(x).$$

In *UncertaintyQuantification.jl* the intervals are also propagated using MADS. In the multivariate case, we can combine two correlated intervals using a Cartesian product

$$[\overline{F}_X^{-1}(\alpha_X), \underline{F}_X^{-1}(\alpha_X)] \times [\overline{F}_Y^{-1}(\alpha_Y), \underline{F}_Y^{-1}(\alpha_Y)],$$

where $(\alpha_X, \alpha_Y) \sim C_{XY}$ are samples of the copula between X and Y.

The reliability analysis can be written as thus:

$$\underline{p}_\text{f} = \int_U \mathbb{I}[\overline{g}(\alpha)] dC, (\alpha) \tag{6}$$

$$\overline{p}_\text{f} = \int_U \mathbb{I}[\underline{g}(\alpha)] dC.(\alpha) \tag{7}$$

In some sense, the two loops from the double-loop method have been reversed, where now the outer-loop handles the random (aleatory) component, and the inner-loop handles the interval propagation (epistemic). Describing the analysis this way essentially gives two separate reliability calculations, with $\underline{g}$ and $\overline{g}$ as the two target performance functions. Rosenblatt transformations may be used to associate a standard normal distribution to the copula C, and one may then use any standard reliability method to compute 6 and 7.

The software implementation is such that this imprecise reliability method can be coupled to any simulation method (FORM, line sampling, etc.) in a straightforward way. As with the double-loop, one can apply different simulations for each bound if desired.

The problem setup for random slicing is identical to that of the double-loop. The only difference is that a `RandomSlicing` instance is passed instead of `DoubleLoop`. Code 4 presents the full function call to solve the example problem presented in

```
x1 = Interval(-1,1,:x1)
x2 = ProbabilityBox{Normal}([Parameter(0,:μ), Interval(1,2,:σ)], :x2)
f = Model(df -> df.x1 .+ df.x2, :f)
```

Code 2: Setup of the example problem. The variable `df` refers to the `DataFrame` that is used internally to store the samples.

```
pf, x_lb, x_ub = probability_of_failure(f, df -> 9 .+ df.f, [x1,x2],
↪  DoubleLoop(FORM()))
```

Code 3: Solve the example problem using double-loop Monte Carlo.

the previous section using random slicing. The lower bound is again estimated using FORM, while we apply subset simulation to obtain the upper bound. The result for the lower bound matches the analytical solution perfectly. The upper bound is estimated accurately as $\overline{p}_f \approx 3.884325e-5$. Note, that in addition to the probability of failure, random slicing also returns other outputs of the underlying simulations, such as the coefficient of variation and the evaluated samples for potential post-processing.

As outlined by Alvarez et al. (2018), other forms of random-sets can in principle be evaluated with this method, such as possibility distributions (Dubois and Prade, 1990) or general Dempster–Shafer structures (Shafer, 1976). However, careful consideration of multivariate extensions of these structures must be taken (Schmelzer, 2023). For this reason, we restrict ourselves to distributions, intervals, and p-boxes for the time being.

6. Numerical Example

In this section we present how to use *UncertaintyQuantification.jl* to solve the *Front Axle* example given in Yuan et al. (2021). Figure 2 presents the cross section of an I-beam profile often used for automobile front axles. The performance function is given as

$$g(x) = \sigma_s - \sqrt{\sigma^2 + 3\tau^2}, \tag{8}$$

where $\sigma_s = 680\,\text{MPa}$ is the yield stress, maximum normal stress is $\sigma = M/W_x$, and the maximum shear stress is $\tau = T/W_\rho$. Here, M and T are the bending moment and torque while W_x and W_ρ are the section factor and polar section factor. These last two are equal to

$$W_x = \frac{a(h-2t)^3}{6h} + \frac{b}{6h}[h^3 - (h-2t)^3] \tag{9}$$

and

$$W_\rho = 0.8bt^2 + 0.4[a^3(h-2t)/t]. \tag{10}$$

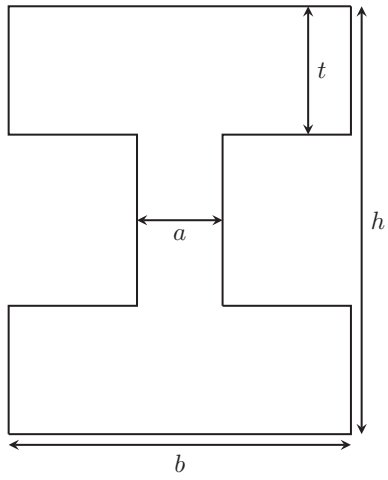

Fig. 2. Cross section of an automobile front axle.

The input variables are assumed to be independent. The mean, standard deviation, and probability distributions are shown in Table 1. Note that, the distributions of a, b, t, and h are truncated to be strictly positive for physical reasons. Code 5 shows how to define these random variables. The function `distribution_parameters` is used here to obtain the parameters of the log-normal distributions for the given mean and standard deviation.

```
subset = SubSetSimulation(2000, 0.1, 10, Normal())
pf, out_lb, out_ub = probability_of_failure(f, df -> 9 .+ df.f, [x1,x2],
↪  RandomSlicing(FORM(), subset))
```

Code 4: Solve the example problem using random slicing. Note the increase in samples for the lower bound.

Table 1. Distributions of the random variables for the numerical example.

Random variable	a (mm)	t (mm)	b (mm)	h (mm)	M (KN ·m)	T (KN ·m)
mean	[11,13]	[13,15]	65	85	3.5	3.1
std	1.2	1.4	6.5	8.5	0.35	0.31
Distribution	Normal	Normal	Normal	Normal	LogNormal	LogNormal

In the next step we define the necessary models. While it is entirely possible to build a single model we have chosen to split Eq. 8 into smaller individual models for improved code readability. In total, we define five `Model` instances, as presented in Code 6.

With the setup now complete we are ready to run the analysis and compute the probability of failure. For comparison we obtain two estimates of the p_f with both the double-loop and random slicing. Internally we apply a standard Monte Carlo simulation with 10^6 for the lower bound and 10^5 samples for the upper bound in the double-loop. Importance sampling with 4000 samples is used for both bounds in random slicing. The code to obtain the solution is presented in Code 7 and the results are shown in Table 2.

Table 2. Results of the front axle example.

Method	$\underline{p}_f$	$\overline{p}_f$
Double-loop	0.000409	0.04272
Random slicing	0.000463	0.042414

7. Third party solvers and HPC

This section promotes the fact that the propagation of imprecise probabilitiesis not restricted to the simple `Model` used throughout this paper, which is essentially a wrapper around a native Julia function. Any model provided by *UncertaintyQuantification.jl* can be used and users can potentially implement their own models which will automatically possess imprecise capabilities. Most importantly, the `ExternalModel` is able to interface with external third party solvers. As such, users have the ability to perform an imprecise reliability analysis using a complex finite element model (FEM) in their preferred solver, such as *Abaqus* or *MOOSE*. Any solver that uses plain text files for input and output can be connected.

High fidelity models come with a significant computational demand on their own. However, this demand is significantly increased when performing an imprecise analysis, which itself is a challenging problem, to a point where the time requirements are no longer feasible. To reduce the required time one can make use of high performance computing (HPC) systems, in this case *Slurm*. This can be done by configuring a `SlurmInterface` and passing it to the `ExternalModel`. For more information and concrete examples we refer to the online documentation and the demo files included in the software.

8. Conclusion

This paper presents the latest development in *UncertaintyQuantification.jl*, a generalized Julia package for uncertainty quantification. Most notably we introduced the new capabilities for performing reliability analyses with imprecise input variables such as intervals or probability boxes. Two alternative methods, the standard double-loop and a technique based on random-set theory have been presented

```
a = ProbabilityBox{Normal}([Interval(11, 13, :μ), Parameter(1.2, :σ)], :a,
↪  0, Inf)
t = ProbabilityBox{Normal}([Interval(13, 15, :μ), Parameter(1.4, :σ)], :t,
↪  0, Inf)

b = RandomVariable(truncated(Normal(65, 6.5), 0, Inf), :b)
h = RandomVariable(truncated(Normal(85, 8.5), 0, Inf), :h)
μ_M, σ_M = distribution_parameters(3.5, 0.35, LogNormal)
M = RandomVariable(LogNormal(μ_M, σ_M), :M)
μ_T, σ_T = distribution_parameters(3.1, 0.31, LogNormal)
T = RandomVariable(LogNormal(μ_T, σ_T), :T)
```

Code 5: Definition of the random variables for the front axle example.

```
function wx(a, h, t, b)
    return a * (h - 2 * t)^3 / (6 * h) + b / (6 * h) * (h^3 - (h - 2 *
    ↪  t)^3)
end

function wz(a, h, t, b)
    return 0.8 * b * t^2 + 0.4 * (a^3 * (h - 2 * t) / t)
end

Wx = Model(df -> wx.(df.a, df.h, df.t, df.b) .* 10^(-9), :Wx)
Wz = Model(df -> wz.(df.a, df.h, df.t, df.b) .* 10^(-9), :Wz)
σ = Model(df -> df.M .* 10^3 ./ df.Wx, :σ)
τ = Model(df -> df.T .* 10^3 ./ df.Wz, :τ)

p = Model(df -> sqrt.(df.σ .^ 2 .+ 3 .* df.τ .^ 2), :p)
```

Code 6: Definition of the models for the front axle example.

and validated using a simple toy example. A more complex real world example was also solved using the software.

Ongoing development is focused on improving efficiency of the propagation of imprecise inputs. The goal is to offer faster alternatives to the current application based on MADS. One promising approach is the combination of Gaussian process surrogate models with Bayesian optimisation. In addition, we are working on implementations of advanced algorithms such as Non-intrusive Imprecise Stochastic Simulation (NISS) (Wei et al., 2019) and its extension Collaborative and Adaptive Bayesian optimisation (CABO) (Hong et al., 2023).

Contributions from other researchers are welcome and strongly encouraged.

Acknowledgement

We would like to acknowledge the invaluable contributions of the following researchers and developers in no particular order: Edoardo Patelli, Andrea Perin, Max Luttmann, Lukas Fritsch, Laurenz Knipper, Jan Grashorn, Thomas Potthast, Felix Mett.

References

Abramson, M. A., C. Audet, J. E. Dennis, and S. L. Digabel (2009). OrthoMADS: A Deterministic MADS Instance with Orthogonal Directions. *20*(2), 948–966.

Alvarez, D. A., F. Uribe, and J. E. Hurtado (2018). Estimation of the lower and upper bounds on the probability of failure using subset simulation

```
σ_s = 680 * 10^6

inputs = [a, t, b, h, M, T]
models = [Wx, Wz, σ, τ, p]
performance = df -> σ_s .- df.p

dl, x_lb, x_ub = DoubleLoop(MonteCarlo(10^6), DoubleLoop(10^5))
pf_dl = probability_of_failure(models, performance, inputs, dl)

rs = RandomSlicing(ImportanceSampling(4000))
pf_rs, out_lb, out_ub = probability_of_failure(models, performance, inputs,
↪  rs)
```

Code 7: Compute the probability of failure of the front axle.

and random set theory. *Mechanical Systems and Signal Processing 100*, 782–801.

Baudin, M., A. Dutfoy, B. Iooss, and A.-L. Popelin (2017). Openturns: an industrial software for uncertainty quantification in simulation. In *Handbook of uncertainty quantification*, pp. 2001–2038. Springer.

Behrensdorf, J., A. Gray, A. Perin, M. Luttmann, M. Broggi, G. Agarwal, L. Fritsch, F. Mett, and L. Knipper (2024, November). Friesischscott/uncertaintyquantification.jl: v0.11.0.

Bezanson, J., A. Edelman, S. Karpinski, and V. B. Shah (2017). Julia: A Fresh Approach to Numerical Computing. *SIAM Review 59*(1), 65–98.

Dubois, D. and H. Prade (1990). Consonant approximations of belief functions. *International Journal of Approximate Reasoning 4*(5-6), 419–449.

Ferson, S., V. Kreinovick, L. Ginzburg, F. Sentz, and D. Meyers (2015). Constructing Probability Boxes and Dempster-Shafer Structures.

Hong, F., P. Wei, J. Song, M. A. Valdebenito, M. G. R. Faes, and M. Beer (2023). Collaborative and Adaptive Bayesian Optimization for bounding variances and probabilities under hybrid uncertainties. *417*, 116410.

Marelli, S. and B. Sudret (2014). UQLab: A Framework for Uncertainty Quantification in Matlab. In *Vulnerability, Uncertainty, and Risk*, pp. 2554–2563. American Society of Civil Engineers.

Olivier, A., D. G. Giovanis, B. S. Aakash, M. Chauhan, L. Vandanapu, and M. D. Shields (2020). UQpy: A general purpose Python package and development environment for uncertainty quantification. *Journal of Computational Science 47*, 101204.

Patelli, E., M. Broggi, M. d. Angelis, and M. Beer (2014). Opencossan: An efficient open tool for dealing with epistemic and aleatory uncertainties. In *Vulnerability, Uncertainty, and Risk: Quantification, Mitigation, and Management*, pp. 2564–2573.

Schmelzer, B. (2023). Random sets, copulas and related sets of probability measures. *International Journal of Approximate Reasoning 160*, 108952.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton university press.

Wei, P., J. Song, S. Bi, M. Broggi, M. Beer, Z. Lu, and Z. Yue (2019). Non-intrusive stochastic analysis with parameterized imprecise probability models: II. Reliability and rare events analysis. *126*, 227–247.

Wright, D. W., R. A. Richardson, W. Edeling, J. Lakhlili, R. C. Sinclair, V. Jancauskas, D. Suleimenova, B. Bosak, M. Kulczewski, T. Piontek, et al. (2020). Building confidence in simulation: applications of easyvvuq. *Advanced Theory and Simulations 3*(8), 1900246.

Yuan, X., M. G. Faes, S. Liu, M. A. Valdebenito, and M. Beer (2021). Efficient imprecise reliability analysis using the Augmented Space Integral. *210*, 107477.