ETH zürich

UQ[py]Lab User Manual Structural Reliability (Rare Event Estimation)

S. Marelli, R. Schöbi, B. Sudret

Chair of Risk, Safety and Uncertainty Quantification
Stefano-Franscini-Platz 5
CH-8093 Zürich

Risk, Safety & Uncertainty Quantification

How to cite UQ[PY]LAB

C. Lataniotis, S. Marelli, B. Sudret, Uncertainty Quantification in the cloud with UQCloud Proceedings of the 4th International Conference on Uncertainty Quantification in Computational Sciences and Engineering (UNCECOMP 2021), Athens, Greece, June 27–30, 2021.

How to cite this manual

S. Marelli, R. Schöbi, B. Sudret, UQ[py]Lab user manual – Structural reliability (Rare event estimation), Report # UQ[py]Lab -V0.9-107, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland, 2023

BIBTEX entry

```
@TechReport{UQdoc_09_107,
author = {Marelli, S. and Sch\"obi, R. and Sudret, B.},
title = {{UQ[py]Lab user manual -- Structural reliability (Rare event estimation)
}},
institution = {Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich,
Switzerland},
year = {2023},
note = {Report \#  UQ[py]Lab -V0.9-107}
}
```

List of contributors:

Name	Contribution
A. Hlobilová	Translation from the UQLab manual

Document Data Sheet

Document Ref.	UQ[PY]LAB-V0.9-107
Title:	UQ[PY]LAB user manual – Structural reliability (Rare event estimation)
Authors:	S. Marelli, R. Schöbi, B. Sudret Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland
Date:	22/3/2023

Doc. Version	Date	Comments
V0.9	22/03/2023	Initial release

Abstract

Structural reliability methods aim at the assessment of the probability of failure of complex systems due to uncertainties associated to their design, manifacturing, environmental and operating conditions. The name *structural reliability* comes from the emergence of such computational methods back in the mid 70's to evaluate the reliability of civil engineering structures. As these probabilities are usually small (*e.g.* $10^{-2} - 10^{-8}$), this type of problems is also known as *rare events estimation* in the recent statistics literature.
The structural reliability module of UQ[PY]LAB offers a comprehensive set of techniques for the efficient estimation of the failure probability of a wide range of systems. Classical (crude Monte Carlo simulation, FORM/SORM, Subset Simulation) and state-of-the-art algorithms (AK-MCS) are available and can be easily deployed in association with other UQ[PY]LAB tools, *e.g.* surrogate modelling or sensitivity analysis.
The structural reliability user manual is divided in three parts:

- A short introduction to the main concepts and techniques used to solve structural reliability problems, with a selection of references to the relevant literature
- A detailed example-based guide, with the explanation of most of the available options and methods
- A comprehensive reference list detailing all the available functionalities in the UQ[PY]LAB structural reliability module.

Keywords: Structural Reliability, FORM, SORM, Importance Sampling, Monte Carlo Simulation, Subset Simulation, AK-MCS, UQ[PY]LAB, rare event estimation

Contents

Chapter 1

Theory

1.1 Introduction

A structural system is defined as a structure required to provide specific functionality under well-defined safety constraints. Such constraints need to be taken into account during the system design phase in view of the expected environmental/operating loads it will be subject to.
In the presence of uncertainties in the physical properties of the system (*e.g.* due to tolerances in the manufacturing), in the environmental loads (*e.g.* due to exceptional weather conditions), or in the operating conditions (*e.g.* traffic), it can occur that the structure operates outside of its nominal range. In such cases, the system encounters a *failure*.
Structural reliability analysis deals with the quantitative assessment of the probability of occurrence of such failures (probability of failure), given a model of the uncertainty in the structural, environmental and load parameters.
Following the formalism introduced in Sudret (2007), this chapter is intended as a brief theoretical introduction and literature review of the available tools in the structural reliability module of UQ[PY]LAB. Consistently with the overall design philosophy of UQ[PY]LAB, all the algorithms presented follow a *black-box* approach, *i.e.* they rely on the point-by-point evaluation of a computational model, without knowledge about its inner structure.

1.2 Problem statement

1.2.1 Limit-state function

A *limit state* can be defined as a state beyond which a system no longer satisfies some performance measure (*ISO Norm 2394*). Regardless on the choice of the specific criterion, a state beyond the limit state is classified as a *failure* of the system.
Consider a system whose state is represented by a random vector of variables $\boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}} \subset \mathbb{R}^M$. One can define two domains $\mathcal{D}_s, \mathcal{D}_f \subset \mathcal{D}_{\boldsymbol{X}}$ that correspond to the *safe* and *failure* regions of the state space $\mathcal{D}_{\boldsymbol{X}}$, respectively. In other words, the system is failing if the current state $\boldsymbol{x} \in \mathcal{D}_f$ and it is operating safely if $\boldsymbol{x} \in \mathcal{D}_s$. This classification makes it possible to construct a *limit-state function* $g(\boldsymbol{X})$ (sometimes also referred to as *performance function*) that assumes

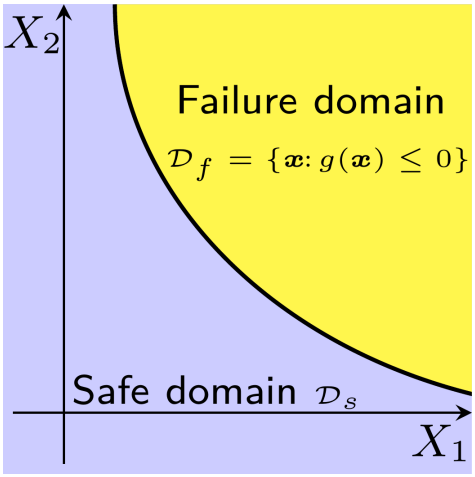

Figure 1: Schematic representation of the safe and failure domains $\mathcal{D}_s$ and $\mathcal{D}_f$ and the corresponding limit-state surface $g(\boldsymbol{x}) = 0$.

positive values in the safe domain and negative values in the failure domain:

$$\begin{aligned} \boldsymbol{x} \in \mathcal{D}_s &\iff g(\boldsymbol{x}) > 0 \\ \boldsymbol{x} \in \mathcal{D}_f &\iff g(\boldsymbol{x}) \leq 0 \end{aligned} \tag{1.1}$$

The hypersurface in M dimensions defined by $g(\boldsymbol{x}) = 0$ is known as the *limit-state surface*, and it represents the boundary between safe and failure domains. A graphical representation of $\mathcal{D}_s, \mathcal{D}_f$ and the corresponding limit-state surface $g(\boldsymbol{x}) = 0$ is given in Figure 1.

1.2.2 Failure Probability

If the random vector of state variables $\boldsymbol{X}$ is described by a joint probability density function (PDF) $\boldsymbol{X} \sim f_{\boldsymbol{X}}(\boldsymbol{x})$, then one can define the *failure probability* P_f as:

$$P_f = \mathbb{P}\left(g(\boldsymbol{X}) \leq 0\right). \tag{1.2}$$

This is the probability that the system is in a failed state given the uncertainties of the state parameters. The failure probability P_f is then calculated as follows:

$$P_f = \int_{\mathcal{D}_f} f_{\boldsymbol{X}}(\boldsymbol{x}) d\boldsymbol{x} = \int_{\{\boldsymbol{x}\colon\, g(\boldsymbol{x}) \leq 0\}} f_{\boldsymbol{X}}(\boldsymbol{x}) d\boldsymbol{x}. \tag{1.3}$$

Note that the integration domain in Eq. (1.3) is only implicitly defined by Eq. (1.1), hence making its direct estimation practically impossible in the general case. This limitation can be circumvented by introducing the *indicator function of the failure domain*, a simple classifier given by:

$$\mathbf{1}_{\mathcal{D}_f}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } g(\boldsymbol{x}) \leq 0 \\ 0 & \text{if } g(\boldsymbol{x}) > 0 \end{cases}, \ \boldsymbol{x} \in \mathcal{D}_{\boldsymbol{X}}.$$

In other words, $\mathbf{1}_{D_f}(\boldsymbol{x}) = 1$ when the input parameters $\boldsymbol{x}$ cause the system to fail and

$\mathbf{1}_{D_f}(\boldsymbol{x}) = 0$ otherwise. This function allows one to cast Eq. (1.3) as follows:

$$P_f = \int_{\mathcal{D}_{\boldsymbol{X}}} \mathbf{1}_{D_f}(\boldsymbol{x}) f_{\boldsymbol{X}}(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}\left[\mathbf{1}_{\mathcal{D}_f}(\boldsymbol{X})\right], \tag{1.4}$$

where $\mathbb{E}[\cdot]$ is the expectation operator with respect to the PDF $f_{\boldsymbol{X}}(\boldsymbol{x})$. This reduces the calculation of P_f to the estimation of the expectation value of $\mathbf{1}_{\mathcal{D}_f}(\boldsymbol{X})$.

1.3 Strategies for the estimation of P_f

From the definition of $\mathbf{1}_{\mathcal{D}_f}(\boldsymbol{x})$ in Section 1.2.2 it is clear that determining whether a certain state vector $\boldsymbol{x} \in \mathcal{D}_{\boldsymbol{X}}$ belongs to $\mathcal{D}_s$ or $\mathcal{D}_f$ requires the evaluation of the limit-state function $g(\boldsymbol{x})$. In the general case this operation can be computationally expensive, *e.g.* when it entails the evaluation of a computational model on the vector $\boldsymbol{x}$. For a detailed overview of standard structural reliability methods and applications, see *e.g.* Ditlevsen and Madsen (1996); Melchers (1999); Lemaire (2009).
In the following, three strategies are discussed for the evaluation of P_f, namely approximation, simulation and adaptive surrogate-modelling-based methods.

Approximation methods

Approximation methods are based on approximating the limit-state function locally at a reference point (*e.g.* with a linear or quadratic Taylor expansion). This class of methods can be very efficient (in that only a relatively small number of model evaluations is needed to calculate P_f), but it tends to become unreliable in the presence of complex, non-linear limit-state functions. Two approximation methods are currently available in UQ[PY]LAB:

- *FORM (First Order Reliability Method)* – it is based on the combination of an iterative gradient-based search of the so-called *design point* and a local linear approximation of the limit-state function in a suitably transformed probabilistic space.
- *SORM (Second Order Reliability Method)* – it is a second-order refinement of the solution of FORM. The computational costs associated to this refinement increase rapidly with the number of input random variables M.

Simulation methods

Simulation methods are based on sampling the joint distribution of the state variables $\boldsymbol{X}$ and using sample-based estimates of the integral in Eq. (1.4). At the cost of being computationally very expensive, they generally have a well-characterized convergence behaviour that can be exploited to calculate confidence bounds on the resulting P_f estimates. Three sampling-based algorithms are available in UQ[PY]LAB:

- *Monte Carlo simulation* – it is based on the direct sample-based estimation of the expectation value in Eq. (1.4). The total costs increase very rapidly with decreasing values of the probability P_f to be computed.

- *Importance Sampling* – it is based on improving the efficiency of Monte Carlo simulation by changing the sampling density so as to favour points in the failure domain $\mathcal{D}_f$. The choice of the importance sampling (a.k.a. instrumental) density generally uses FORM results.

- *Subset Simulation* – it is based on iteratively solving and combining a sequence of conditional reliability analyses by means of Markov Chain Monte Carlo (MCMC) simulation.

Metamodel-based adaptive methods

Metamodel-based adaptive methods are based on iteratively building surrogate models that approximate the limit-state function in the direct vicinity of the limit-state surface. The metamodels (see *e.g.* UQ[PY]LAB User Manual – Polynomial Chaos Expansions and UQ[PY]LAB User Manual – Kriging (Gaussian process modelling)) are adaptively refined by adding limit-state function evaluations to their experimental designs until a suitable convergence criterion related to the accuracy of P_f is satisfied. One algorithm is currently available in UQ[PY]LAB, namely *Adaptive Kriging Monte Carlo Simulation (AK-MCS)*. It is based on building a Kriging (aka Gaussian process regression) surrogate model from a small initial sampling of the input vector $\boldsymbol{X}$. The surrogate is then iteratively refined close to the currently estimated limit-state surface so as to evaluate accurately the probability of failure.

In the following, a detailed description of each of the methods is given.

1.4 Approximation methods

1.4.1 First Order Reliability Method (FORM)

The first order reliability method aims at the approximation of the integral in Eq. (1.3) with a three-step approach:

- An *isoprobabilistic transform* of the input random vector $\boldsymbol{X} \sim f_{\boldsymbol{X}}(\boldsymbol{x})$ into a standard normal vector $\boldsymbol{U} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_M)$

- A search for the most likely failure point in the standard normal space (SNS), known as the *design point* $\boldsymbol{U}^*$

- A linearization of the limit-state surface at the design point $\boldsymbol{U}^*$ and the analytical computation of the resulting approximation of P_f.

1.4.1.1 Isoprobabilistic transform

The first step of the FORM method is to transform the input random vector $\boldsymbol{X} \sim f_{\boldsymbol{X}}$ into a standard normal vector $\boldsymbol{U} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_M)$. The corresponding isoprobabilistic transform $\mathcal{T}$ reads:

$$\boldsymbol{X} = \mathcal{T}^{-1}(\boldsymbol{U}) \tag{1.5}$$

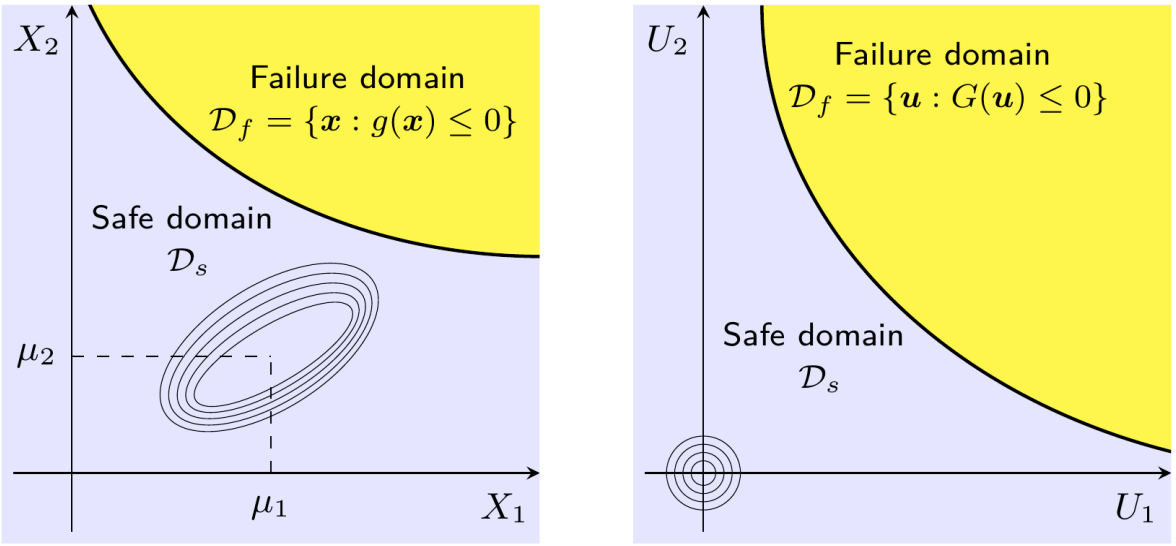

Figure 2: Graphical representation of the isoprobabilistic transform from physical to standard normal space in Eq. (1.5). From Sudret, 2015: Lectures on structural reliability and risk analysis.

For details about the available isoprobabilistic transforms in UQ[PY]LAB, please refer to the UQ[PY]LAB User Manual – the INPUT module .

This transform can be used to map the integral in Eq. (1.3) from the physical space of $\boldsymbol{X}$ to the standard normal space of $\boldsymbol{U}$:

$$P_f = \int_{\mathcal{D}_f} f_{\boldsymbol{X}}(\boldsymbol{x}) d\boldsymbol{x} = \int_{\{\boldsymbol{u} \in \mathbb{R}^M \,:\, G(\boldsymbol{u}) \leq 0\}} \varphi_M(\boldsymbol{u}) d\boldsymbol{u} \tag{1.6}$$

where $G(\boldsymbol{u}) = g(\mathcal{T}^{-1}(\boldsymbol{u}))$ is the limit-state function evaluated in the standard normal space and $\varphi_M(\boldsymbol{u})$ is the standard multivariate normal PDF given by:

$$\varphi_M(\boldsymbol{u}) = (2\pi)^{-M/2} \exp\left(-\frac{1}{2}(u_1^2 + \cdots + u_M^2)\right). \tag{1.7}$$

A graphical illustration of the effects of this transform for a simple 2-dimensional case is given in Figure 2. The advantage of casting the problem in the standard normal space is that it is a probability space equipped with the Gaussian probability measure $\mathbb{P}_G$:

$$\mathbb{P}_G(\boldsymbol{U} \in A) = \int_A \varphi_M(\boldsymbol{u}) d\boldsymbol{u} = \int_A (2\pi)^{-M/2} \exp\left(u_1^2 + \cdots + u_M^2\right) d\boldsymbol{u}. \tag{1.8}$$

This probability measure is spherically symmetric: $\varphi_M(\boldsymbol{u})$ only depends on $\|\boldsymbol{u}\|^2$ and it decays exponentially as $\varphi_M(\boldsymbol{u}) \sim \exp\left(-\|\boldsymbol{u}\|^2/2\right)$. Therefore, when evaluating the integral in Eq. (1.6) in the standard normal space, most of the contributions are given by the region *closest to the origin*. The FORM method capitalizes on this property by linearly approximating the limit-state surface in the region closest to the origin of the standard normal space.

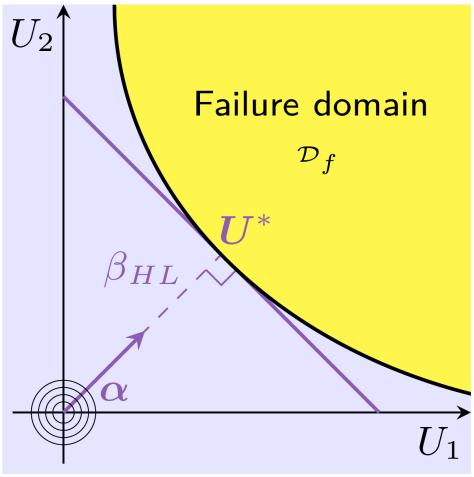

Figure 3: Graphical representation of the linearization of the limit-state function around the design point at the basis of the FORM estimation of P_f. From Sudret, 2015: Lectures on structural reliability and risk analysis.

1.4.1.2 Search for the design point

The *design point* $\boldsymbol{U}^*$ is defined as the point in the failure domain closest to the origin of the standard normal space:

$$\boldsymbol{U}^* = \underset{\boldsymbol{u}\in\mathbb{R}^M}{\operatorname{argmin}} \left\{ \|\boldsymbol{u}\|, G(\boldsymbol{u}) \leq 0 \right\}. \tag{1.9}$$

Due to the probability measure in Eq. (1.8), $\boldsymbol{U}^*$ can be interpreted as the most likely failure point in the standard normal space. The norm of the design point $\|\boldsymbol{U}^*\|$ is an important quantity in structural reliability known as the *Hasofer-Lind reliability index* (Hasofer and Lind, 1974):

$$\beta_{HL} = \|\boldsymbol{U}^*\|. \tag{1.10}$$

An important property of the β_{HL} index is that it is directly related to the *exact* failure probability P_f in the case of linear limit-state function in the standard normal space:

$$P_f = \Phi(-\beta_{HL}), \tag{1.11}$$

where Φ is the standard normal cumulative density function. The estimation of P_f in the FORM algorithm is based on approximating the limit-state function as the hyperplane tangent to the limit-state function at the design point. Figure 3 illustrates this approximation graphically for the two-dimensional case.

In the general non-linear case, Eq. (1.9) may be cast as a constrained optimization problem with Lagrangian:

$$\mathcal{L}(\boldsymbol{u}, \lambda) = \frac{1}{2}\|\boldsymbol{u}\|^2 + \lambda G(\boldsymbol{u}) \tag{1.12}$$

where λ is the Lagrange multiplier. The related optimality conditions read:

$$\begin{aligned} \nabla_{\boldsymbol{u}} \mathcal{L}(\boldsymbol{U}^*, \lambda^*) = 0, \\ \frac{\partial \mathcal{L}}{\partial \lambda}(\boldsymbol{U}^*, \lambda^*) = 0, \end{aligned} \tag{1.13}$$

which can be explicitly written as:

$$\begin{aligned} G(\boldsymbol{U}^*) = 0, \\ \boldsymbol{U}^* + \lambda^* \nabla G(\boldsymbol{U}^*) = 0. \end{aligned} \tag{1.14}$$

The first condition in Eq. (1.14) guarantees that the design point belongs to the limit-state surface. The second condition guarantees that the vector $\boldsymbol{U}^*$ is colinear to the limit-state surface normal vector at $\boldsymbol{U}^*$, *i.e.* $\nabla G(\boldsymbol{U}^*)$. The standard iterative approach to solve this non-linear constrained optimization problem is given by the *Rackwitz-Fiessler algorithm* (Rackwitz and Fiessler, 1978).

Hasofer-Lind - Rackwitz-Fiessler algorithm (HL-RF)

The rationale behind the Rackwitz-Fiessler algorithm is to iteratively solve a linearized problem around the current point. Normally, the algorithm is started with $\boldsymbol{U}_0 = \boldsymbol{0}$.
At each iteration, the limit-state function is approximated as:

$$G(\boldsymbol{U}) \approx G(\boldsymbol{U}_k) + \nabla G_{|\boldsymbol{U}_k} \cdot (\boldsymbol{U} - \boldsymbol{U}_k) \tag{1.15}$$

The two optimality conditions in Eq. (1.14) read for each iteration k:

$$\begin{aligned} \nabla G_{|\boldsymbol{U}_k} \cdot (\boldsymbol{U}_{k+1} - \boldsymbol{U}_k) + G(\boldsymbol{U}_k) = 0 \\ \boldsymbol{U}_{k+1} = \lambda \nabla G_{|\boldsymbol{U}_k}, \end{aligned} \tag{1.16}$$

which after some basic algebra reduce to:

$$\boldsymbol{U}_{k+1} = \frac{\nabla G_{|\boldsymbol{U}_k} \cdot \boldsymbol{U}_k - G(\boldsymbol{U}_k)}{\|\nabla G_{|\boldsymbol{U}_k}\|^2} \nabla G_{|\boldsymbol{U}_k}. \tag{1.17}$$

By introducing the unit vector:

$$\boldsymbol{\alpha}_k = -\frac{\nabla G_{|\boldsymbol{U}_k}}{\|\nabla G_{|\boldsymbol{U}_k}\|}, \tag{1.18}$$

one finally obtains:

$$\boldsymbol{U}_{k+1} = \left[\boldsymbol{\alpha}_k \cdot \boldsymbol{U}_k + \frac{G(\boldsymbol{U}_k)}{\|\nabla G_{|\boldsymbol{U}_k}\|} \right] \boldsymbol{\alpha_k}. \tag{1.19}$$

The associated estimate of the reliability index β_k associated to the k-th iteration is then:

$$\beta_k = \boldsymbol{\alpha_k} \cdot \boldsymbol{U}_k + \frac{G(\boldsymbol{U}_k)}{\|\nabla G_{|\boldsymbol{U}_k}\|}. \tag{1.20}$$

Perfect convergence of the algorithm is obtained when $G(\boldsymbol{U}^*) = 0$, yielding $\beta_{HL} = \boldsymbol{\alpha^*} \cdot \boldsymbol{U}^*$.
However, in practice the algorithm is iterated until some stopping criteria are satisfied, i.e.,

Table 1: Common stopping criteria for the FORM algorithm and associated description.

Criterion	Typical value	Description
$\lvert\beta_{k+1} - \beta_k\rvert \leq \epsilon_\beta$	10^{-3}	Stability of β between iterations
$\lVert\boldsymbol{U}_{k+1} - \boldsymbol{U}_k\rVert \leq \epsilon_U$	10^{-3}	Stability of $\boldsymbol{U}$ between iterations
$\lvert G(\boldsymbol{U}_{k+1})/G(\boldsymbol{U}_0)\rvert \leq \epsilon_G$	10^{-6}	Closeness to the limit-state surface

until one or more convergence conditions are verified. The standard stopping criteria used in FORM are reported in Table 1.

Note: In UQ[PY]LAB the gradients $\nabla G(\boldsymbol{U}_k)$ in Eqs. (1.13) to (1.20) are calculated numerically in the standard normal space and not in the physical space.

Improved HL-RF algorithm (iHL-RF)

The Rackwitz-Fiessler algorithm is a particular case of a wide class of iterative algorithms generically denoted as *descent direction algorithms*, of the form:

$$\boldsymbol{U}_{k+1} = \boldsymbol{U}_k + \lambda_k \boldsymbol{d}_k, \tag{1.21}$$

where λ_k is the *step size* at the k-th iteration and $\boldsymbol{d}_k$ is the corresponding *descent direction* given by:

$$\boldsymbol{d}_k = \frac{\nabla G_{|\boldsymbol{U}_k} \cdot \boldsymbol{U}_k - G(\boldsymbol{U}_k)}{\lVert\nabla G_{|\boldsymbol{U}_k}\rVert^2} \nabla G_{|\boldsymbol{U}_k} - \boldsymbol{U}_k. \tag{1.22}$$

In the original HL-RF algorithm, $\lambda_k = 1 \quad \forall k$. Zhang and Der Kiureghian (1995) proposed an "improved" version of the same algorithm that takes advantage of a more sophisticated step-size calculation based on the assumption that $G(\boldsymbol{U})$ is differentiable everywhere. They introduced the *merit function* $m(\boldsymbol{U})$:

$$m(\boldsymbol{U}) = \frac{1}{2}\lVert\boldsymbol{U}\rVert + c\lvert G(\boldsymbol{U})\rvert, \tag{1.23}$$

where $c > \dfrac{\lVert\boldsymbol{U}\rVert}{\lVert\nabla G(\boldsymbol{U})\rVert}$ is a real penalty parameter. This function has its global minimum in the same location as the original Eq. (1.9), as well as the same descent direction $\boldsymbol{d}$. In addition, it allows one to use the Armijo rule (Zhang and Der Kiureghian, 1995) to determine the best step length λ_k at each iteration as:

$$\lambda_k = \max_s \{b^s \mid m(\boldsymbol{U}_k + b^s \boldsymbol{d}_k) - m(\boldsymbol{U}_k) \leq -ab^s \nabla m(\boldsymbol{U}_k) \cdot \boldsymbol{d}_k\}, \tag{1.24}$$

where $a, b \in (0,1)$ are pre-selected parameters, and $s \in \mathbb{N}$.

1.4.1.3 FORM results

Once the design point $\boldsymbol{U}^*$ is identified, it can be used to extract additional important information. According to Eq. (1.20), after the convergence of FORM the Hasofer-Lind index β_{HL}

is given by:

$$\beta_{HL} = \boldsymbol{\alpha^*} \cdot \boldsymbol{U}^*, \tag{1.25}$$

with associated failure probability:

$$P_{f,\text{FORM}} = \Phi^{-1}(\beta_{HL}) \tag{1.26}$$

The local sensitivity indices S_i are defined as the fraction of the variance of the safety margin $g(\boldsymbol{X}) = G(\boldsymbol{U})$ due to the component of the design vector $\boldsymbol{U}_i$. It can be demonstrated that they are given by:

$$S_i = \left(\frac{\partial G}{\partial u_i}\bigg|_{\boldsymbol{U}^*} \right)^2 / \|\nabla G(\boldsymbol{U}^*)\|^2. \tag{1.27}$$

From Eq. (1.18) it follows that:

$$S_i = \alpha_i^2. \tag{1.28}$$

If the input variables are independent, then each coordinate in the SNS U_i corresponds to a single input variable in the physical space X_i. Therefore, the importance factor of each X_i is identified with α_i^2.

1.4.2 Second Order Reliability Method (SORM)

The second-order reliability method (SORM) is a second-order refinement of the FORM P_f estimate. After the design point $\boldsymbol{U}^*$ is identified by FORM, the failure probability is approximated by a tangent hyperparaboloid defined by the second order Taylor expansion of $G(\boldsymbol{U}^*)$ given by:

$$G(\boldsymbol{U}) \approx \nabla G^T_{|\boldsymbol{U}^*} \cdot (\boldsymbol{U} - \boldsymbol{U}^*) + \frac{1}{2}(\boldsymbol{U} - \boldsymbol{U}^*)^T \boldsymbol{H} (\boldsymbol{U} - \boldsymbol{U}^*), \tag{1.29}$$

where $\boldsymbol{H}$ is the Hessian matrix of the second derivatives of $G(\boldsymbol{U})$ evaluated at $\boldsymbol{U}^*$.

The failure probability in the SORM approximation can be written as a correction factor of the FORM estimate that depends on the curvatures of the hyper-hyperboloid in Eq. (1.29). To estimate the curvatures, the hyperparaboloid in Eq. (1.29) is first cast in canonical form by rotating the coordinates system such that one of its axes is the $\boldsymbol{\alpha}$ vector. Usually the last coordinate is chosen arbitrarily for this purpose. A rotation matrix $\boldsymbol{Q}$ can be built by setting $\boldsymbol{\alpha}$ as its last row and by using the Gram-Schmidt procedure to orthogonalize the remaining components of the basis. $\boldsymbol{Q}$ is a square matrix such that $\boldsymbol{Q}^\mathsf{T}\boldsymbol{Q} = \boldsymbol{I}$. The resulting vector $\boldsymbol{V}$ satisfies:

$$\boldsymbol{U} = \boldsymbol{Q}\boldsymbol{V}. \tag{1.30}$$

In the new coordinates system and after some basic algebra (see *e.g.* Breitung (1989) and Cai and Elishakoff (1994)), one can rewrite Eq. (1.29) as:

$$G(\boldsymbol{V}) \approx \|\nabla G(\boldsymbol{U}^*)\|(\beta - V_M) + \frac{1}{2}(\boldsymbol{V} - \boldsymbol{V}^*)\boldsymbol{Q}\boldsymbol{H}\boldsymbol{Q}^\mathsf{T}(\boldsymbol{V} - \boldsymbol{V}^*) \tag{1.31}$$

where $\boldsymbol{\beta}$ is the Hasofer-Lind reliability index calculated by FORM, $V_M \overset{\text{def}}{=} \boldsymbol{\alpha}^\mathsf{T}(\boldsymbol{Q}^\mathsf{T}\boldsymbol{V})$ and

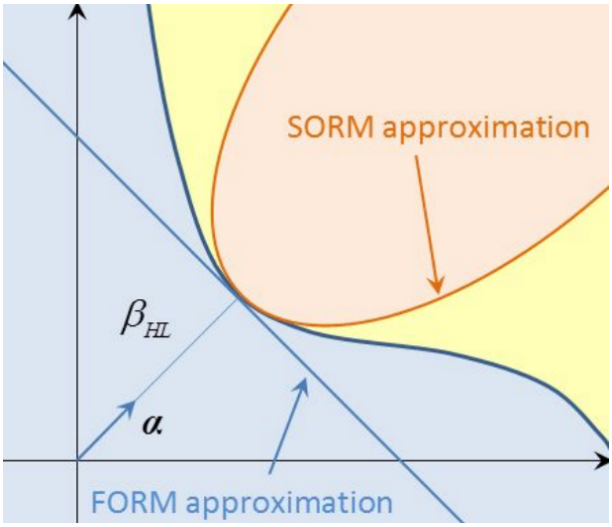

Figure 4: Comparison between FORM and SORM approximations of the failure domain for a simple 2-dimensional case. From Sudret, 2015: Lectures on structural reliability and risk analysis.

$\boldsymbol{V}^* = \{0, \cdots, \beta\}^\mathsf{T}$ is the design point in the new coordinates system. By dividing Eq. (1.31) by the gradient norm $\|\nabla G(\boldsymbol{U}^*)\|$ and introducing the matrix $\boldsymbol{A} \overset{\text{def}}{=} \boldsymbol{Q}\boldsymbol{H}\boldsymbol{Q}^\mathsf{T}/\|\nabla G(\boldsymbol{U}^*)\|$, one obtains:

$$\tilde{G}(\boldsymbol{V}) \approx \beta - V_M + \frac{1}{2}(\boldsymbol{V} - \boldsymbol{V}^*)\boldsymbol{A}(\boldsymbol{V} - \boldsymbol{V}^*), \tag{1.32}$$

where $\tilde{G}(\boldsymbol{V}) = G(\boldsymbol{V})/\|\nabla G(\boldsymbol{U}^*)\|$. After neglecting second-order terms in V_M and diagonalizating the $\boldsymbol{A}$ matrix via eigenvalue decomposition one can rewrite Eq. (1.32) explicitly in terms of the curvatures κ_i of an hyper-paraboloid with axis $\boldsymbol{\alpha}$:

$$\tilde{G}(\boldsymbol{V}) \approx \beta - V_M + \frac{1}{2}\sum_{1}^{M-1} \kappa_i V_i. \tag{1.33}$$

For small curvatures $\kappa_i < 1$, the failure probability P_f can be approximated by the Breitung formula (Breitung, 1989):

$$P^B_{f,\mathsf{SORM}} = \Phi(-\beta_{HL}) \prod_{i=1}^{M-1} (1 + \beta_{HL}\, \kappa_i)^{-\frac{1}{2}} \quad \kappa_i < 1 \tag{1.34}$$

Note that for small curvatures the Breitung formula approaches the FORM linear limit. The accuracy of Eq. (1.34) decreases for larger values of κ_i, sometimes even if $\kappa_i < 1$ (Cai and Elishakoff, 1994). A more accurate formula is given by the Hohenbichler formula (Hohenbichler et al., 1987):

$$P^H_{f,\mathsf{SORM}} = \Phi(-\beta_{HL}) \prod_{i=1}^{M-1} \left(1 + \frac{\varphi(\beta_{HL})}{\Phi(-\beta_{HL})}\kappa_i\right)^{-\frac{1}{2}}. \tag{1.35}$$

Additional methods are available in the literature for the exact computation of the failure probability, *e.g.*Tvedt (1990). They are, however, outside the scope of this manual.

1.5 Simulation methods

1.5.1 Monte Carlo Simulation

Monte Carlo (MC) simulation is used to directly compute the integral in Eq. (1.4) by sampling the probabilistic input model. Given a sample of size N of the input random vector $\boldsymbol{X}$, $\mathcal{X} = \left\{\boldsymbol{x}^{(1)}, \dots \boldsymbol{x}^{(N)}\right\}$, the unbiased MCS estimator of the expectation value in Eq. (1.4) is given by:

$$P_{f,\text{MC}} \overset{\text{def}}{=} \widehat{P}_f = \frac{1}{N}\sum_{k=1}^{N} \mathbf{1}_{\mathcal{D}_f}(\boldsymbol{x}^{(k)}) = \frac{N_{fail}}{N}, \tag{1.36}$$

where N_{fail} is the number of samples such that $g(\boldsymbol{x}) \leq 0$. In other words, the Monte Carlo estimate of the failure probability is the fraction of samples that belong to the failure domain over the total number of samples. An advantage of Monte Carlo simulation is that it provides an error estimate for Eq. (1.36). Indeed the indicator function $\mathbf{1}_{\mathcal{D}_f}(\boldsymbol{x})$ follows by construction a Bernoulli distribution with mean $\mu_{\mathbf{1}_{\mathcal{D}_f}} = P_f$ and variance $\sigma^2_{\mathbf{1}_{\mathcal{D}_f}} = P_f(1-P_f)$. For large enough N it can be approximated by the normal distribution:

$$\widehat{P}_f \sim \mathcal{N}\left(\widehat{\mu}_{\mathbf{1}_{\mathcal{D}_f}}, \widehat{\sigma}_{\mathbf{1}_{\mathcal{D}_f}}\right), \tag{1.37}$$

where $\widehat{\mu}_{\mathbf{1}_{\mathcal{D}_f}} = \widehat{P}_f$ and $\widehat{\sigma}_{\mathbf{1}_{\mathcal{D}_f}} = \sqrt{\widehat{P}_f(1-\widehat{P}_f)}$. Hence, the estimator of P_f has a normal distribution with mean $\widehat{P}_f$ and variance given by:

$$\widehat{\sigma}^2_{P_f} = \frac{\sigma^2_{\mathbf{1}_{\mathcal{D}_f}}}{N} = \frac{\widehat{P}_f(1-\widehat{P}_f)}{N}. \tag{1.38}$$

Confidence intervals on $\widehat{P}_f$ can therefore be given as follows (Rubinstein, 1981):

$$\widehat{P}_f \in \left[\widehat{P}_f^- \overset{\text{def}}{=} \widehat{P}_f + \widehat{\sigma}_{P_f}\Phi^{-1}(\alpha/2),\ \widehat{P}_f^+ \overset{\text{def}}{=} \widehat{P}_f + \widehat{\sigma}_{P_f}\Phi^{-1}(1-\alpha/2)\right], \tag{1.39}$$

where $\Phi(x)$ is the standard normal CDF and $\alpha \in [0,1]$ is a scalar such that the calculated bounds correspond to a confidence level of $1-\alpha$. An important measure for assessing the convergence of a MCS estimator is given by the coefficient of variation CoV defined as:

$$CoV = \frac{\sigma_{\widehat{P}_f}}{\widehat{P}_f} = \sqrt{\frac{1-\widehat{P}_f}{N\widehat{P}_f}}. \tag{1.40}$$

The coefficient of variation of the MCS estimate of a failure probability therefore decreases with $\sqrt{N}$ and increases with decreasing P_f. To give an example, to estimate a $P_f = 10^{-3}$ with 10% accuracy $N = 10^5$ samples are needed. The CoV is often used as a convergence criterion to adaptively increase the MC sample size until some desired CoV is reached.

An *associated generalized reliability index* β_{MCS} can be defined as:

$$\beta_{\text{MCS}} = -\Phi^{-1}(\widehat{P}_f). \tag{1.41}$$

In analogy, upper and lower confidence bounds on β_{MCS} can be directly inferred from the confidence bounds on $\widehat{P}_f$ in Eq. (1.39):

$$\beta^{\pm}_{\text{MCS}} = -\Phi^{-1}(\widehat{P}^{\pm}_f). \tag{1.42}$$

The MCS method is powerful, when applicable, due to its statistically sound formulation and global convergence. However, its main drawback is the relatively slow converge rate that depends strongly on the probability of failure.

1.5.2 Importance Sampling

Importance sampling (IS) is an extension of the FORM and MCS methods that combines the fast convergence of FORM with the robustness of MC. The basic idea is to recast Eq. (1.4) as:

$$P_f = \int_{\mathcal{D}_{\boldsymbol{X}}} \mathbf{1}_{D_f}(\boldsymbol{x}) \frac{f_{\boldsymbol{X}}(\boldsymbol{x})}{\Psi(\boldsymbol{x})} \Psi(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{\Psi}\left[\mathbf{1}_{\mathcal{D}_f}(\boldsymbol{X}) \frac{f_{\boldsymbol{X}}(\boldsymbol{X})}{\Psi(\boldsymbol{X})}\right], \tag{1.43}$$

where $\Psi(\boldsymbol{X})$ is an $M-$dimensional *sampling distribution* (also referred to as *importance distribution*) and $\mathbb{E}_{\Psi}$ denotes the expectation value with respect to the same distribution. The estimate of P_f given a sample $\mathcal{X} = \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\right\}$ drawn from Ψ is therefore given by:

$$P_{f,\text{IS}} = \frac{1}{N} \sum_{k=1}^{N} \mathbf{1}_{\mathcal{D}_f}(\boldsymbol{x}^{(k)}) \frac{f_{\boldsymbol{X}}(\boldsymbol{x}^{(k)})}{\Psi(\boldsymbol{x}^{(k)})}. \tag{1.44}$$

In the standard normal space, Eq. (1.43) can be rewritten as:

$$P_f = \mathbb{E}_{\Psi}\left[\mathbf{1}_{\mathcal{D}_f}(\mathcal{T}^{-1}(\boldsymbol{U})) \frac{\varphi_M(\boldsymbol{U})}{\Psi(\boldsymbol{U})}\right]. \tag{1.45}$$

When the results from a previous FORM analysis are available, a particularly efficient sampling distribution in the standard normal space is given by (Melchers, 1999):

$$\Psi(\boldsymbol{u}) = \varphi_M(\boldsymbol{u} - \boldsymbol{U}^*) \tag{1.46}$$

where $\boldsymbol{U}^*$ is the estimated design point. Given a sample $\mathcal{U} = \left\{\boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(N)}\right\}$ of $\Psi(\boldsymbol{u})$, the estimate of P_f becomes:

$$P_{f,\text{IS}} = \frac{1}{N} \exp\left(-\beta^2_{HL}/2\right) \sum_{k=1}^{N} \mathbf{1}_{\mathcal{D}_f}\left(\mathcal{T}^{-1}(\boldsymbol{u}^{(k)})\right) \exp\left(-\boldsymbol{u}^{(k)} \cdot \boldsymbol{U}^*\right) \tag{1.47}$$

with corresponding variance:

$$\widehat{\sigma}^2_{P_{f,\text{IS}}} = \frac{1}{N} \frac{1}{N-1} \sum_{k=1}^{N} \left(\mathbf{1}_{\mathcal{D}_f}\left(\mathcal{T}^{-1}(\boldsymbol{u}^{(k)})\right) \frac{\varphi(\boldsymbol{u}^{(k)})}{\Psi(\boldsymbol{u}^{(k)})} - P_{f,\text{IS}}\right)^2. \tag{1.48}$$

The coefficient of variation and the confidence bounds $P^{\pm}_{f,\text{IS}}$ can be calculated analogously to Eqs. (1.40) and (1.39), respectively, and can be used as a convergence criterion to adaptively

improve the estimation of $P_{f,\text{IS}}$. The corresponding generalized reliability index reads:

$$\beta_{\text{IS}} = -\Phi^{-1}(\widehat{P}_{f,\text{IS}}), \tag{1.49}$$

with upper and lower bounds:

$$\beta^{\pm}_{\text{IS}} = -\Phi^{-1}(\widehat{P}^{\pm}_{f,\text{IS}}). \tag{1.50}$$

Note that exact convergence of FORM is not necessary to obtain accurate results, even an approximate *sampling distribution* can significantly improve the convergence rate compared to standard MC sampling.

1.5.3 Subset Simulation

Monte Carlo simulation may require a large number of limit-state function evaluations to converge with an acceptable level of accuracy when P_f is small (see Eq. (1.40)). Subset simulation is a technique introduced by Au and Beck (2001) that aims at offsetting this limitation by solving a series of simpler reliability problems with intermediate failure thresholds.
Consider a sequence of failure domains $\mathcal{D}_1 \supset \mathcal{D}_2 \supset \cdots \supset \mathcal{D}_m = \mathcal{D}_f$ such that $\mathcal{D}_f = \bigcap_{k=1}^{m} \mathcal{D}_k$. With the conventional definition of limit-state function in Eq. (1.1), such sequence can be built with a series of decreasing failure thresholds $t_1 > \cdots > t_m = 0$ and the corresponding intermediate failure domains $\mathcal{D}_k = \{\boldsymbol{x} : g(\boldsymbol{x}) \leq t_k\}$. One can then combine the probability mass of each intermediate failure region by means of conditional probability. By introducing the notation $\mathbb{P}(\mathcal{D}_{\boldsymbol{X}}) = \mathbb{P}(\boldsymbol{x} \in \mathcal{D}_{\boldsymbol{X}})$ one can write (Au and Beck, 2001):

$$P_f = \mathbb{P}(\mathcal{D}_m) = P\left(\bigcap_{k=1}^{m} \mathbb{P}(\mathcal{D}_k)\right) = \mathbb{P}(\mathcal{D}_1) \prod_{i=1}^{m-1} \mathbb{P}(\mathcal{D}_{i+1}|\mathcal{D}_i)\,. \tag{1.51}$$

With an appropriate choice of the intermediate thresholds $t_1, \ldots, t_m$, Eq. (1.51) can be evaluated as a series of structural reliability problems with relatively high probabilities of failure that are then solved with MC simulation. In practice the intermediate probability thresholds t_i are chosen on-the-fly such that they correspond to intermediate values $\mathbb{P}(\mathcal{D}_k) \approx 0.1$. The convergence of each intermediate estimation is therefore much faster than the direct search for P_f given in Eq. (1.36).

1.5.3.1 Sampling

To estimate P_f from Eq. (1.51) one thus needs to estimate the intermediate probabilities $\mathbb{P}(\mathcal{D}_1)$ and conditional probabilities $\{\mathbb{P}(\mathcal{D}_{i+1}|\mathcal{D}_i)\,, i = 1, \ldots, m-1\}$. Given an initial threshold t_1, $\mathbb{P}(\mathcal{D}_1)$ can be readily estimated from a sample of size N_S of the input distribution $\mathcal{X} = \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\right\}$ with Eq. (1.36):

$$\mathbb{P}(\mathcal{D}_1) \approx \widehat{P}_1 = \frac{1}{N_S} \sum_{k=1}^{N_S} \mathbf{1}_{\mathcal{D}_1}(\boldsymbol{x}^{(k)}). \tag{1.52}$$

The remaining conditional probabilities can be estimated similarly, but an efficient sampling

algorithm is needed for the underlying conditional distributions. The latter can be efficiently accomplished by using the modified Metropolis-Hastings Markov Chain Monte Carlo (MCMC) simulation introduced by Au and Beck (2001).

1.5.3.2 Intermediate failure thresholds

The efficiency of the subset-simulation method depends on the choice of the intermediate failure thresholds t_k. If the thresholds are too large the MCS convergence in each subset would be very good, but the number of subsets needed would increase. Vice-versa, too small intermediate thresholds would correspond to fewer subsets with inaccurate estimates of the underying $P(\mathcal{D}_k)$. A strategy to deal with this problem comes by sampling each subset $\mathcal{D}_k$ and determining each threshold t_k as the empirical quantile that correspond to a predetermined failure probability, typically $\mathbb{P}(\mathcal{D}_k) \approx P_0 = 0.1$. Note that for practical reasons, P_0 is normally limited to $0 < P_0 \leq 0.5$. For each subset, the samples falling below the calculated threshold are used as MCS seeds for the next subset (Au and Beck, 2001).

1.5.3.3 Subset simulation algorithm

The subset simulation algorithm can be summarized in the following steps:

1. Sample the original space with standard MC sampling (see UQ[PY]LAB User Manual – the INPUT module for efficient sampling strategies)
2. Calculate the empirical quantile t_k in the current subset such that $\widehat{P}_k \approx P_0$
3. Using the samples below the identified quantile as the seeds of parallel MCMC chains, sample $\mathcal{D}_{k+1}|\mathcal{D}_k$ until a predetermined number of samples is available
4. Repeat Steps 2 and 3 until the identified quantile $t_m < 0$
5. Calculate the failure probability of the last subset $\widehat{P}_m$ by setting $t_m = 0$
6. Combine the intermediate calculated failure probabilities into the final estimate of $\widehat{P}_f$.

The last step of the algorithm consists simply in evaluating Eq. (1.51) with the current estimates of the conditional probabilities P_i:

$$\widehat{P}_{f,\text{SS}} = \prod_{i=1}^{m} \widehat{P}_i = P_0^{m-1} \widehat{P}_m. \tag{1.53}$$

1.5.3.4 Error estimation

Due to the intrinsic correlation of the samples drawn from each subset resulting from the MCMC sampling strategy, the estimation of a CoV for the P_f estimate in Eq. (1.53) is non-trivial. Au and Beck (2001) and Papaioannou et al. (2015) derived an estimate for the CoV of $\widehat{P}_f$:

$$CoV_f \approx \sum_{i=1}^{m} \delta_i^2, \tag{1.54}$$

where m is the number of subsets and δ_i is defined as:

$$\delta_i = \sqrt{\frac{1-P_i}{NP_i}\,(1+\gamma_i)}, \tag{1.55}$$

with

$$\gamma_i = 2\sum_{k=1}^{N/N_s}\left(1-\frac{kN_s}{N}\right)\rho_i(k), \tag{1.56}$$

where N_S is the number of seeds, P_i is the conditional failure probability of the $i-$th subset and $\rho_i(k)$ is the average k-lag auto-correlation coefficient of the Markov Chain samples in the $i-$th subset. By assuming normally distributed errors, confidence bounds $P^{\pm}_{f,\text{SS}}$ can be given on $P_{f,\text{SS}}$ based on the calculated CoV_f in analogy with Eq. (1.39). The corresponding generalized reliability index reads:

$$\beta_{\text{SS}} = -\Phi^{-1}(\widehat{P}_{f,\text{SS}}), \tag{1.57}$$

with upper and lower bounds:

$$\beta^{\pm}_{\text{SS}} = -\Phi^{-1}(\widehat{P}^{\pm}_{f,\text{SS}}). \tag{1.58}$$

1.6 Metamodel-based methods

1.6.1 Adaptive Kriging Monte Carlo Simulation

Adaptive Kriging Monte Carlo Simulation (AK-MCS) combines Monte Carlo simulation with adaptively built Kriging (a.k.a. Gaussian process modelling) metamodels. In cases where the evaluation of the limit-state function is costly, Monte Carlo simulation and its variants may become intractable due to the large number of limit-state function evaluations they require. In AK-MCS, a Kriging metamodel surrogates the limit-state function to reduce the total computational costs of the Monte Carlo simulation.

Kriging metamodels (see UQ[PY]LAB User Manual – Kriging (Gaussian process modelling)) predict the value of the limit-state function most accurately in the vicinity of the experimental design samples $\mathcal{X} = \left\{\boldsymbol{X}^{(1)},\dots,\boldsymbol{X}^{(n)}\right\}$. These samples, however, are generally not optimal to estimate the failure probability. Thus, an adaptive experimental design algorithm is introduced to increase the accuracy of the surrogate model in the vicinity the limit-state function. This is achieved by adding carefully selected samples to the experimental design of the Kriging metamodel based on the current estimate of the limit-state surface ($g(\boldsymbol{x}) = 0$).

The adaptive experimental design algorithm is summarized as follows (Echard et al., 2011; Schöbi et al., 2016):

1. Generate a small initial experimental design $\mathcal{X} = \left\{\boldsymbol{x}^{(1)},\dots,\boldsymbol{x}^{(N_0)}\right\}$ and evaluate the corresponding limit-state function responses $\mathcal{Y} = \left\{y^{(1)},\dots,y^{(N_0)}\right\} = \left\{g(\boldsymbol{x}^{(1)}),\dots,g(\boldsymbol{x}^{(N_0)})\right\}$

2. Train a Kriging metamodel $\widehat{g}$ based on the experimental design $\{\mathcal{X},\mathcal{Y}\}$

3. Generate a large set of N_{MC} candidate samples $\mathcal{S} = \left\{\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(N_{MC})}\right\}$ and predict the corresponding metamodel responses $\left\{\widehat{g}(\boldsymbol{s}^{(1)}), \ldots, \widehat{g}(\boldsymbol{s}^{(N_{MC})})\right\}$

4. Choose the best next sample $\boldsymbol{s}^*$ to be added to the experimental design $\mathcal{X}$ based on an appropriate learning function

5. Check whether some convergence criterion is met. If it is, skip to Step 7, otherwise continue with Step 6

6. Add $\boldsymbol{s}^*$ and the corresponding limit-state function response $y^* = g(\boldsymbol{s}^*)$ to the experimental design of the metamodel. Return to Step 2

7. Estimate the failure probability through Monte Carlo simulation with the final limit-state function surrogate $\widehat{g}(\boldsymbol{x})$.

1.6.1.1 Selection of the best next candidate sample

A learning function is a measure of the attractiveness of a candidate sample $\boldsymbol{X}$ with respect to improving the estimate of the failure probability when it is added to the experimental design $\mathcal{X}$. A variety of learning functions are available in the literature (Bichon et al., 2008; Dani et al., 2008; Echard et al., 2011; Srinivas et al., 2012; Ginsbourger et al., 2013; Dubourg, 2011), amongst which the U-function (Echard et al., 2011). The U-function is based on the concept of misclassification and is defined for a Gaussian process as follows:

$$U(\boldsymbol{X}) = \frac{\left|\mu_{\widehat{g}}(\boldsymbol{X})\right|}{\sigma_{\widehat{g}}(\boldsymbol{X})}, \tag{1.59}$$

where $\mu_{\widehat{g}}(\boldsymbol{X})$ and $\sigma_{\widehat{g}}(\boldsymbol{X})$ are the prediction mean and standard deviation of $\widehat{g}$. A misclassification happens when the sign of the surrogate model and the sign of the underlying limit-state function do not match. The corresponding probability of misclassification is then:

$$P_m(\boldsymbol{X}) = \Phi\left(-U(\boldsymbol{X})\right),$$

where Φ is the CDF of a standard Gaussian variable.

The next candidate sample from the set $\mathcal{S} = \left\{\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(N_{MC})}\right\}$ is chosen as the one that maximizes the probability of misclassification or, in other words, as the one most likely to have been misclassified as safe/failed by the surrogate limit-state function $\widehat{g}(\boldsymbol{x})$:

$$\boldsymbol{s}^* = \arg\min_{\boldsymbol{s}\in\mathcal{S}} U(\boldsymbol{s}) \equiv \arg\max_{\boldsymbol{s}\in\mathcal{S}} P_m(\boldsymbol{s}). \tag{1.60}$$

Another popular learning function is the *expected feasibility function* (EFF) (Bichon et al.,

2008):

$$EFF(\boldsymbol{x}) = \mu_{\widehat{g}}(\boldsymbol{x}) \left[2\Phi\left(\frac{-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \Phi\left(\frac{-\epsilon - \mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \Phi\left(\frac{\epsilon - \mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right)\right] - \sigma_{\widehat{g}}(\boldsymbol{x}) \left[2\varphi\left(\frac{-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \varphi\left(\frac{-\epsilon - \mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \varphi\left(\frac{\epsilon - \mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right)\right] + \epsilon \left[\Phi\left(\frac{\epsilon - \mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \Phi\left(\frac{-\epsilon - \mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right)\right], \quad (1.61)$$

where $\epsilon = 2\sigma_{\widehat{g}}(\boldsymbol{x})$ and φ is the PDF value of a standard normal Gaussian variable. The next candidate sample is then chosen by:

$$\boldsymbol{s}^* = \arg\max_{\boldsymbol{s}\in\mathcal{S}} EFF(\boldsymbol{s}). \quad (1.62)$$

1.6.1.2 Convergence criteria

The convergence criterion terminates the addition of samples to the experimental design of the Kriging metamodel and thus terminates the improvement in the accuracy of the failure probability estimate. The standard convergence criterion related to the U-function is defined as follows (Echard et al., 2011): the iterations stop when $\min_i U(\boldsymbol{s}^{(i)}) > 2$ where $i = 1, \ldots, N_{MC}$. Schöbi et al. (2016) demonstrated that this criterion is very conservative and that an alternative stopping criterion, related to the uncertainty in the estimate of the failure probability itself, is often more efficient in the context of structural reliability. It is given by the following condition:

$$\frac{\widehat{P}_f^+ - \widehat{P}_f^-}{\widehat{P}_f^0} \leq \epsilon_{\widehat{P}_f}, \quad (1.63)$$

where $\epsilon_{\widehat{P}_f} = 10\%$ and the three failure probabilities are defined as:

$$\widehat{P}_f^0 = \mathbb{P}\left(\mu_{\widehat{g}}(\boldsymbol{X}) \leq 0\right), \quad (1.64)$$

$$\widehat{P}_f^{\pm} = \mathbb{P}\left(\mu_{\widehat{g}}(\boldsymbol{X}) \mp k\sigma_{\widehat{g}}(\boldsymbol{X}) \leq 0\right), \quad (1.65)$$

where $k = \Phi^{-1}(1 - \alpha/2)$ sets the confidence level $(1 - \alpha)$, typically $k = \Phi^{-1}(97.5\%) = 1.96$.

A similar convergence criterion can be defined for the reliability index:

$$\frac{\widehat{\beta}^+ - \widehat{\beta}^-}{\widehat{\beta}^0} \leq \epsilon_{\widehat{\beta}}, \quad (1.66)$$

where the threshold $\epsilon_{\widehat{\beta}_f} = 5\%$ and the three reliability indices correspond to the aforementioned failure probabilities:

$$\widehat{\beta}^0 = -\Phi^{-1}(\widehat{P}_f^0), \quad (1.67)$$

$$\widehat{\beta}^{\pm} = -\Phi^{-1}(\widehat{P}_f^{\mp}). \quad (1.68)$$

1.6.1.3 AK-MCS with a PC-Kriging metamodel

As originally proposed by Echard et al. (2011), AK-MCS uses an ordinary Kriging model for approximating the limit-state function. As demonstrated by Schöbi et al. (2016) replacing the ordinary Kriging metamodel with a Polynomial-Chaos-Kriging (PC-Kriging) one (see also UQ[PY]LAB User Manual – PC-Kriging) can significantly improve the convergence of the algorithm. The corresponding reliability methods is called *Adaptive PC-Kriging Monte Carlo Simulation* (APCK-MCS) and is available in UQ[PY]LAB as an advanced option of AK-MCS (see Section 2.3.6.2).

Chapter 2

Usage

In this section, a reference problem will be set up to showcase how each of the techniques in Chapter 1 can be deployed in UQ[PY]LAB.

2.1 Reference problem: R-S

The benchmark of choice to showcase the methods described in Section 1.3 is a basic problem in structural reliability, namely the R-S case. It is one of the simplest possible abstract setting consisting of only two input state variables: a resistance R and a stress S. The system fails when the stress is higher than the resistance, leading to the following limit-state function:

$$\boldsymbol{X} = \{R, S\} \qquad g(\boldsymbol{X}) = R - S; \tag{2.1}$$

The two-dimensional probabilistic input model consists of independent variables distributed according to Table 2.

Table 2: Distributions of the input parameters of the $R - S$ model in Eq. (2.1).

Name	Distributions	Parameters	Description
R	Gaussian	[5, 0.8]	Resistance of the system
S	Gaussian	[2, 0.6]	Stress applied to the system

An example UQ[PY]LAB script that showcases how to deploy all of the algorithms available in the structural reliability module can be found in the example file:

`1-RS.py`

2.2 Problem set-up

Solving a structural reliability problem in UQ[PY]LAB requires the definition of three basic components:

- a MODEL object that describes the limit-state function
- an INPUT object that describes the probabilistic model of the random vector $\boldsymbol{X}$

- a reliability ANALYSIS object.

The UQ[PY]LAB framework is first initialized with the following command:

```
from uqpylab import sessions

# Start the session
mySession = sessions.cloud()
# (Optional) Get a convenient handle to the command line interface
uq = mySession.cli
# Reset the session
mySession.reset()
```

The model in Eq. (2.1) can be added as a MODEL object directly with a `Python` vectorized string as follows:

```
MOpts = {
    'Type': 'Model',
    'mString': 'X(:,1) - X(:,2)',
    'isVectorized': 1
    }
myModel = uq.createModel(MOpts)
```

For more details on the available options to create a model object in UQ[PY]LAB, please refer to the UQ[PY]LAB User Manual – the MODEL module.
Correspondingly, an INPUT object with independent Gaussian variables as specified in Table 2 can be created as:

```
IOpts = {
    'Marginals': [
        {'Name': 'R',               # Resistance
        'Type': 'Gaussian',
        'Moments': [5.0 , 0.8]
        },
        {'Name': 'S',               # Stress
        'Type': 'Gaussian',
        'Moments': [2.0 , 0.6]
        }
    ]
}

myInput = uq.createInput(IOpts)
```

For more details about the configuration options available for an INPUT object, please refer to the UQ[PY]LAB User Manual – the INPUT module.

2.3 Reliability analysis with different methods

This section showcases how all the methods introduced in Section 1.3, can be deployed in UQ[PY]LAB. In addition, visualization and advanced options are also described in detail. The following methods are showcased in this section:

- FORM: Section 2.3.1
- SORM: Section 2.3.2

- Monte Carlo Simulation: Section 2.3.3
- Importance Sampling: Section 2.3.4
- Subset Simulation: Section 2.3.5
- AK-MCS: Section 2.3.6

2.3.1 First Order Reliability Method (FORM)

Running a FORM analysis on the specified UQ[PY]LAB MODEL and INPUT objects does not require any specific configuration. The following minimum syntax is required:

```
FORMOpts = {
    'Type': 'Reliability',
    'Method':'FORM'
}
FORMAnalysis = uq.createAnalysis(FORMOpts)
```

Once the analysis is performed, a report with the FORM results can be printed on screen by:

```
uq.print(FORMAnalysis)
```

which produces the following:

```
--------------------------------------------------
FORM
--------------------------------------------------
Pf               1.3499e-03
BetaHL           3.0000
ModelEvaluations 8
--------------------------------------------------
Variables        R           S
Ustar          -2.400000   1.800000
Xstar          3.08e+00    3.08e+00
Importance     0.640000    0.360000
--------------------------------------------------
```

The results can be visualized graphically as follows:

```
uq.display(FORMAnalysis)
```

which produces the images in Figure 5. Note that the graphical representation of the FORM iterations (right panel of Figure 5) is only produced for the 2-dimensional case.

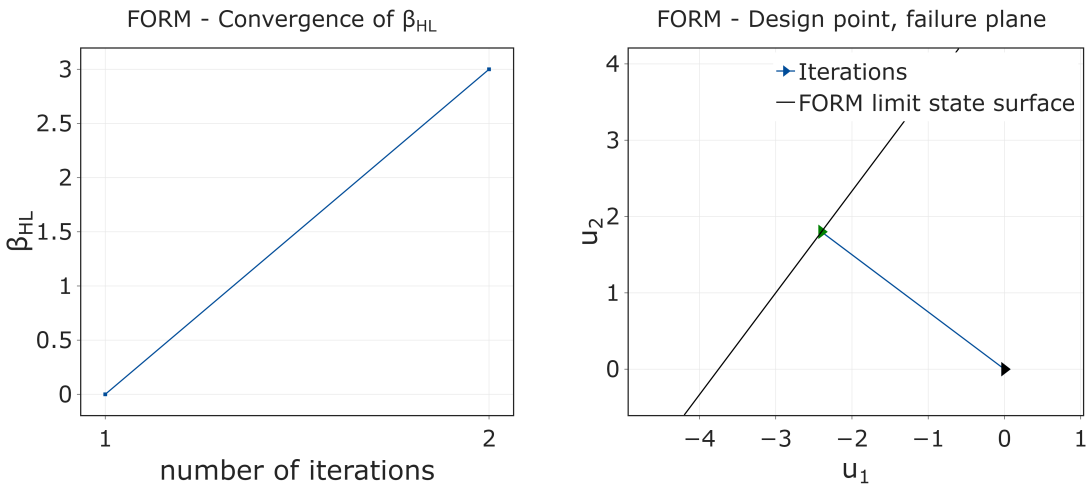

Figure 5: Graphical visualization of the results of the FORM analysis in Section 2.3.1.

Note: In the preceding example no specifications are provided. If not further specified the FORMruns with the following defaults:

- Algorithm to find the design point: `'iHLRF'`;
- Starting point for the Rackwitz-Fiessler (RW) algorithm: $(0, \ldots, 0)$;
- Tolerance value for the RW algorithm on the design point: 10^{-4};
- Tolerance value for the RW algorithm on the limit-state function: 10^{-4};
- Maximum number of iterations for the RW algorithm: 100;
- Failure is defined for: limit-state $g(x) \leq 0$.

Since FORM is a gradient-based method, the gradient of the limit-state function needs to be computed. This is done using finite differences with the following defaults:

- Type of finite difference scheme: `'forward'`;
- Value of the difference scheme: 10^{-3}.

2.3.1.1 Accessing the results

The analysis results can be accessed in the `FORMAnalysis['Results']` dictionary:

```
{
    'BetaHL': 3,
    'Pf': 0.0013,
    'Ustar': [-2.4000 1.8000],
    'Xstar': [3.0800 3.0800],
    'Importance': [0.6400 0.3600],
    'ModelEvaluations': 8,
    'Iterations': 2,
    'History': {...}
}
```

In the `Results` dictionary , `Pf` is the estimate of P_f according to Eq. (1.26), `BetaHL` the corresponding Hasofer-Lindt reliability index, `Ustar` the design point $\boldsymbol{U}^*$ in the standard normal space, `Xstar` the correspondingly transformed design point in the original space $\boldsymbol{X}^* = \mathcal{T}^{-1}(\boldsymbol{U}^*)$, `Importance` the importance factors S_i in Eq. (1.28), `Iterations` the number of FORM iterations needed to converge, `ModelEvaluations` the total number of evaluations of the limit-state function, and `History` a set of additional information about the convergence behaviour of the algorithm.

2.3.1.2 Advanced options

Several advanced options are available for the FORM method to tweak which algorithm is used to calculate the solution. They can be specified by adding a `FORM` key to the FORM options dictionary `FORMOpts`. In the following, the most common advanced options for FORM are specified:

- **Specify the FORM algorihm:** by default, the iHL-RF algorithm is used (see page 8). The original HL-RF algorithm (see page 7) can be enforced by adding:

```
FORMOpts['FORM'] = {'Algorithm': 'HLRF'}
```

- **Specify a starting point:** by default the search of the design point is started in the SNS at $\boldsymbol{U}_0 = \boldsymbol{0}$. It is possible to specify an alternative starting point (useful, *e.g.*, when multiple design points are expected) as:

```
FORMOpts['FORM'] = {'StartingPoint': [u1,...,uM]}
```

  where `[u1,...,uM]` are the desired coordinates of the starting point in the SNS.

- **Numerical calculation of the gradient:** advanced options related to the numerical calculation of the gradient can be specified by using the `FORMOpts['Gradient']` dictionary. As an example, to specify a gradient relative step-size $h = 0.001$ one can write:

```
FORMOpts['Gradient'] = {'h': 0.001}
```

  Details on the gradient computation options are given in Table 7, page 46.

For a comprehensive list of the advanced options available to the FORM method, please see Table 6, page 45.

2.3.2 Second Order Reliability Method (SORM)

A SORM analysis is set up very similarly to its FORM counterpart:

```
SORMOpts = {
    'Type': 'Reliability',
    'Method': 'SORM'
}

SORMAnalysis = uq.createAnalysis(SORMOpts)
```

Once the analysis is performed, a report with the FORM+SORM results can be printed by:

```
uq.print(SORMAnalysis)
```

which produces the following:

```
-----------------------------------------------------
FORM/SORM
-----------------------------------------------------
Pf                1.3499e-03
BetaHL            3.0000
PfFORM            1.3499e-03
PfSORM            1.3499e-03
PfSORMBreitung    1.3499e-03
ModelEvaluations  20
-----------------------------------------------------
Variables         R            S
Ustar          -2.400000    1.800000
Xstar          3.08e+00     3.08e+00
Importance     0.640000     0.360000
-----------------------------------------------------
```

The results can be visualized graphically as follows:

```
uq.display(SORMAnalysis)
```

which produces the same images as in FORM (Figure 5), as SORM is only a refinement of the final FORM P_f estimate.

Note: In the preceding example no specifications are provided. If not further specified the SORM runs with the same defaults as FORM.

2.3.2.1 Accessing the results

The analysis results can be accessed in the `SORMAnalysis['Results']` dictionary:

```
{
    'BetaHL': 3,
    'Pf': 0.0013,
    'Ustar': [-2.4000, 1.8000],
    'Xstar': [3.0800, 3.0800],
    'Importance': [0.6400, 0.3600],
    'ModelEvaluations': 20,
    'Iterations': 2,
    'History': {...},
    'PfSORM': 0.0013,
    'PfSORMBreitung': 0.0013,
    'BetaSORM': 3.0000,
    'BetaSORMBreitung': 3.0000,
    'Curvatures': [-2.8422e-10, 0],
    'PfFORM': 0.0013
}
```

The `Results` dictionary contains the same keys as FORM (see Section 2.3.1) (when necessary with a `FORM` or `SORM` suffix for clarity). In addition, the two `PfSORMBreitung` and `PfSORM` keys provide the $P_{f,\text{SORM}}$ and $P^H_{f,\text{SORM}}$ given in Eqs. (1.34) and (1.35), respectively.

2.3.2.2 Advanced options

The SORM method shares the same advanced options as the FORM method, described in Section 2.3.1.2.

2.3.3 Monte Carlo Simulation (MCS)

The Monte Carlo simulation (MCS) algorithm only requires the user to specify the maximum number of limit-state function evaluations, corresponding to N in Eq. (1.36), if different from the default value $N = 10^5$. As an example, to run a reliability analysis with $N = 10^6$ samples one can write:

```
MCOpts = {
    'Type': 'Reliability',
    'Method': 'MCS'
    'Simulation': {
        'MaxSampleSize': 1e6
    }
}
MCAnalysis = uq.createAnalysis(MCOpts)
```

Once the analysis is performed, a report with the Monte Carlo simulation results can be printed on screen by:

```
uq.print(MCAnalysis)
```

which produces the following:

```
--------------------------------------------------
Monte Carlo simulation
--------------------------------------------------
Pf                1.4000e-03
Beta              2.9889
CoV               0.0267
ModelEvaluations 1000000
PfCI              [1.3267e-03 1.4733e-03]
BetaCI            [2.9733e+00 3.0053e+00]
--------------------------------------------------
```

The results can be visualized graphically as follows:

```
uq.display(MCAnalysis)
```

which produces the convergence plots in Figure 6 and the plot of Monte Carlo sample points in Figure 7. Note that in `uq.display`, the maximum number of samples plotted is $n = 10^5$ to limit the size of the figure.

Note: In the preceding example only the maximum sample size for the analysis is provided. If not further specified the Monte Carlo simulation runs with the following defaults:

- Confidence level: 0.05;
- Number of samples evaluated per batch: 10^4;
- Failure is defined for: limit-state $g(x) \leq 0$.

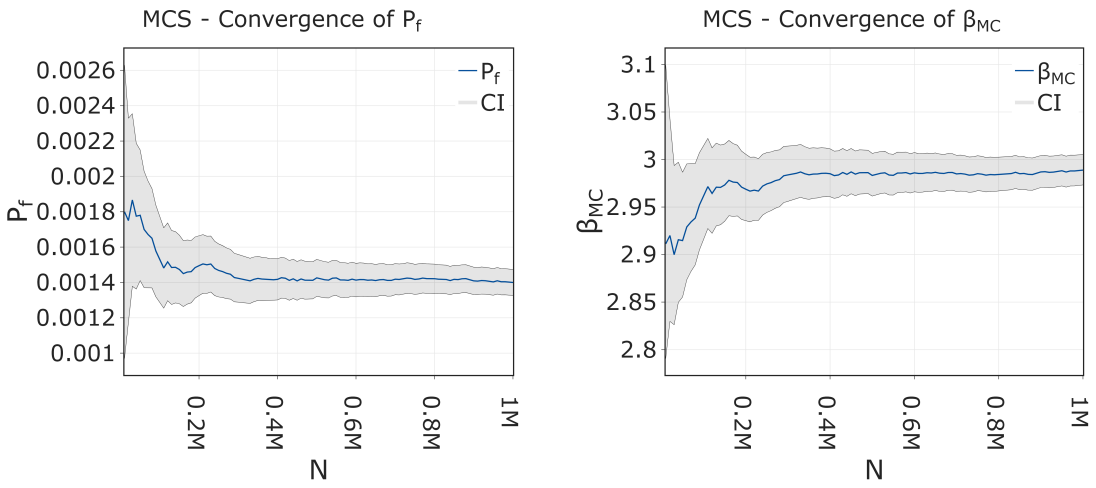

Figure 6: Graphical visualization of the convergence of the Monte Carlo simulation analysis in Section 2.3.3.

2.3.3.1 Accessing the results

The analysis results can be accessed in the `MCAnalysis['Results']` dictionary:

```
{
    'Pf': 0.0014,
    'Beta': 2.9889,
    'CoV': 0.0267,
    'ModelEvaluations': 1000000,
    'PfCI': [0.0013, 0.0015],
    'BetaCI': [2.9733, 3.0053],
    'History': {...},
}
```

The `Results` dictionary contains the following keys : `Pf`, the estimated $\widehat{P}_{f,\mathrm{MC}}$ as in Eq. (1.36); `Beta`, the corresponding generalized reliability index in Eq. (1.41); `CoV`, the coefficient of variation calculated with Eq. (1.40); `ModelEvaluations`, the total number of limit-state function evaluations; `PfCI`, the confidence intervals calculated with Eq. (1.39); `BetaCI`, the corresponding confidence intervals on β_{HL} calculated with Eq. (1.42); `History`, a dictionary containing the convergence of P_f, CoV and the corresponding confidence intervals calculated at preset sample batches (by default once every 10^4 samples). The content of the `History` dictionary is used to produce the convergence plot shown in Figure 6.

2.3.3.2 Advanced options

The advanced options available in Monte Carlo simulation are related to the convergence criterion of the algorithm and to the definition of the confidence bounds reported in the `MCAnalysis['Results']` dictionary. In the following, a list of the most commonly used parameters for a MC analysis are given:

- **Specify a target CoV and a corresponding batch size:** in addition to specifying the

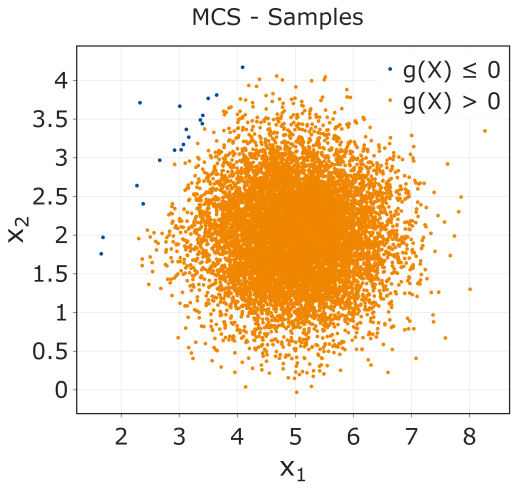

Figure 7: Graphical visualization of the samples of the Monte Carlo simulation analysis in Section 2.3.3.

`MaxSampleSize` option, one can specify a target CoV. The algorithm will sequentially add batches of points to the current sample and stop as soon as the current CoV is below the specify threshold. To specify a target $CoV = 0.01$ and batches of size $N_B = 10^4$, one can write:

```
MCOpts['Simulation'] = {
    'TargetCoV': 0.01,
    'BatchSize': 1e4
}
```

Note that the two options are independent from each other. The `BatchSize` option is also used to set the breakpoints for the `MCAnalysis['Results']['History']` dictionary.

- **Specify α for the confidence intervals:** the α in Eq. (1.39) can also be specified. To set $\alpha = 0.1$, one can write:

```
MCOpts['Simulation'] = {'Alpha': 0.1}
```

For a comprehensive list of the advanced options available for Monte Carlo simulation, please refer to Table 5, page 44.

2.3.4 Importance Sampling

Importance sampling shares configuration options from both the FORM and the Monte Carlo simulation methods. A basic IS analysis can be setup with the following code:

```
ISOpts = {
    'Type': 'Reliability',
    'Method': 'IS'
}
ISAnalysis = uq.createAnalysis(ISOpts)
```

Using this minimal setup the analysis will run FORM first with the default options as in Section 2.3.1 to determine the design point $\boldsymbol{U}^*$. Then the standard normal importance density centred at the obtained design point is used. Sampling is carried out with the following default options: $N = 1000$, batch size $N_B = 100$.

Once the analysis is performed, a report with the importance sampling results can be printed on screen by:

```
uq.print(ISAnalysis)
```

which produces the following:

```
--------------------------------------------------
Importance Sampling
--------------------------------------------------
Pf               1.3549e-03
Beta             2.9989
CoV              0.0590
ModelEvaluations 1008
PfCI             [1.1983e-03 1.5114e-03]
BetaCI           [2.9654e+00 3.0361e+00]
--------------------------------------------------
```

The results can be visualized graphically as follows:

```
uq.display(ISAnalysis)
```

which produces the convergence image in Figure 8. As with FORM, the second panel in Figure 8 is produced only in the 2D case.

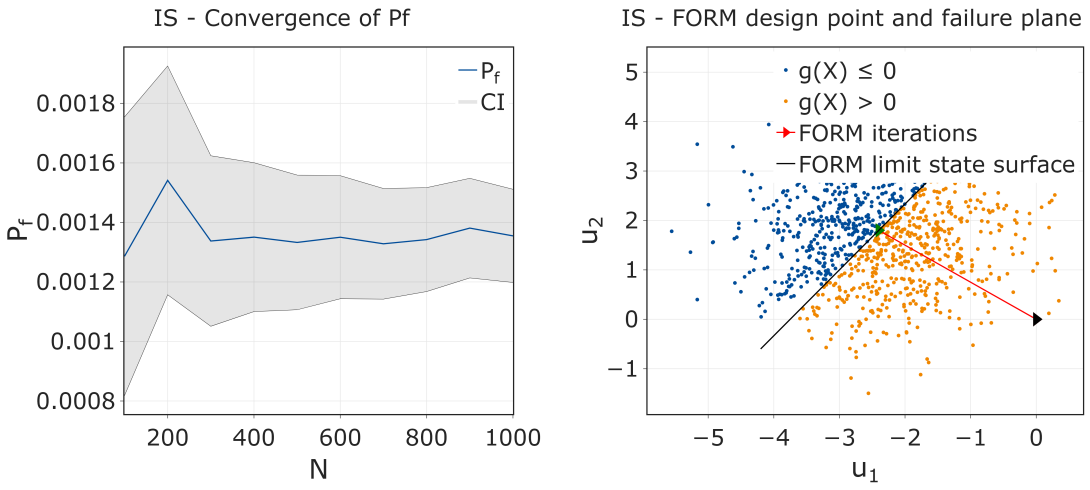

Figure 8: Graphical visualization of the convergence of the importance sampling analysis in Section 2.3.4.

Note: In the preceding example no specifications are provided. The Importance Sampling shares the same defaults values as FORM and MCS. The exceptions are:

- Maximum number of evaluated samples: 10^3;
- Number of samples evaluated per batch: 10^2.

2.3.4.1 Accessing the results

The results of the importance sampling analysis can be accessed with the `ISAnalysis['Results']` dictionary:

```
'Pf': 0.0014,
'Beta': 2.9989,
'CoV': 0.0590,
'PfCI': [0.0012, 0.0015],
'BetaCI': [2.9654, 3.0361],
'ModelEvaluations': 1008,
'History': {...},
'FORM': {...},
```

The basic dictionary of `Results` closely resembles that of Monte Carlo simulation (see Section 2.3.3.1). However, an additional dictionary `Results['FORM']` is available:

```
    'BetaHL': 3,
    'Pf': 0.0013,
    'ModelEvaluations': 8,
    'Ustar': [-2.4000, 1.8000],
    'Xstar': [3.0800, 3.0800],
    'Importance': [0.6400, 0.3600],
    'ModelEvaluations': 8,
    'Iterations': 2,
    'History': {...}
```

This dictionary is identical to the FORM results given in Section 2.3.1.1.

2.3.4.2 Advanced options

The importance sampling algorithm accepts all of the options specific to both FORM and Monte Carlo simulation, described in Section 2.3.1.2 and Section 2.3.3.2. In addition, two additional options can be specified for importance sampling:

- **Specify existing FORM results:** by default, importance sampling first runs FORM to determine the design point, followed by sampling around this design point to calculate $P_{f,\mathrm{IS}}$. If the results of a previous FORM analysis are already available, they can be specified so as to avoid running FORM again. If the results are stored in a `FORMResults` dictionary with the same format as described in Section 2.3.1.1, one can write:

  ```
  ISOpts['IS'] = {'FORM': FORMResults}
  ```

  Alternatively, one can also directly specify a pre-existing UQ[PY]LAB FORM or SORM analysis, say `FORMAnalysis`:

```
ISOpts['IS'] = {'FORM': FORMAnalysis}
```

- **Specify a custom *sampling distribution*:** alternatively, one can directly specify a custom sampling distribution. This can be achieved by providing the marginals and copula structure of the desired distribution, *e.g.* `IOpts` in Section 2.2, as follows:

  ```
  ISOpts['IS'] = {'Instrumental': IOpts}
  ```

  Alternatively, a pre-existent UQ[PY]LAB `INPUT` object, say `myISInput`, can also be specified:

  ```
  ISOpts['IS'] = {'Instrumental': myISInput}
  ```

  In case the model has multiple outputs N_{out}, it might be desirable to specify a custom sampling distribution for each one of them. This can be done by either providing the `IOpts` as a $1 \times N_{out}$ `dictionary` or the pre-existing inputs `myISInputs` as a $1 \times N_{out}$ `uq_input` object. Please note, that custom distributions should be specified either for all outputs or for none.

For a complete overview of the available options specific to the importance sampling algorithm, see Table 8.

2.3.5 Subset Simulation

The subset simulation algorithm can be used with the default options $P_0 = 0.1$ and $N_S = 10^3$ by specifying:

```
SSOpts = {
    'Type': 'Reliability',
    'Method': 'Subset'
}
SSimAnalysis = uq.createAnalysis(SSOpts);
```

Once the analysis is performed, a report with the subset-simulation results can be printed on screen by:

```
uq.print(SSimAnalysis)
```

which produces the following:

```
--------------------------------------------------
Subset simulation
--------------------------------------------------
Pf               1.3300e-03
Beta             3.0045
CoV              0.2520
ModelEvaluations 2689
PfCI             [6.7300e-04 1.9870e-03]
BetaCI           [2.8802e+00 3.2060e+00]
--------------------------------------------------
```

The results can be visualized graphically as follows:

```
uq.display(SSimAnalysis)
```

which illustrates the samples of each subset in Figure 9 (applicable only for one and two-dimensional problems).

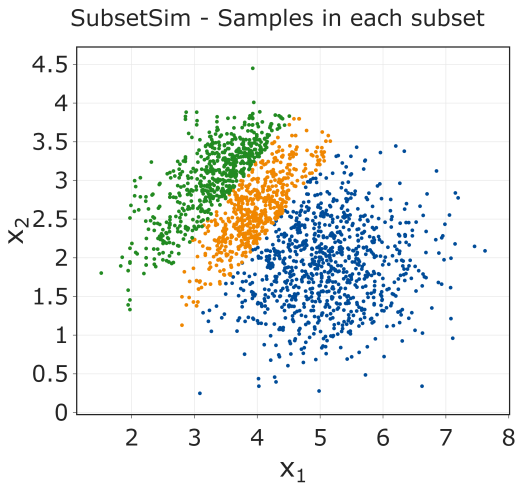

Figure 9: Graphical visualization of the convergence of the subset simulation analysis in Section 2.3.5.

Note: In the preceding example no specifications are provided. Additionally, there are the following default values:

- Target conditional failure probability of auxiliary limit-states: 0.1;
- Maximum number of subsets: 20;
- Type of the proposal distribution in the Markov Chain: `'uniform'`;
- Parameter (standard deviation / halfwidth) of the proposal distribution: 1.

2.3.5.1 Accessing the results

The results of subset simulation are stored in the `SSimAnalysis['Results']` dictionary:

```
{
    'Pf': 0.00133,
    'Beta': 3.0045,
    'History': {...},
    'CoV': 0.2520,
    'ModelEvaluations': 2689,
    'NumberSubsets': 3,
    'PfCI': [0.0006730, 0.001987],
    'BetaCI': [2.8802, 3.2060]
}
```

The keys in the `Results` dictionary have the same meaning as their counterparts in importance sampling and Monte Carlo simulation. Further, the key `NumberSubsets` denotes the number of subsets. Note that the `ModelEvaluations` key does not contain exactly the

expected $N = N_S * m * (1 - P_0) = 2700$ limit-state function evaluations, but a slightly smaller $N = 2689$. This discrepancy is due to the modified Metropolis-Hastings MCMC acceptance criterion described in Au and Beck (2001), which in some uncommon cases can reject samples without the need of evaluating the limit-state function.

2.3.5.2 Advanced options

Subset simulation uses the same advanced options as Monte Carlo simulation described in Section 2.3.3.2, as well as some additional options. The most important are summarized in the following:

- **Specify P_0:** the value of P_0 in Eq. (1.53) can be specified in $0 < P_0 \leq 0.5$. One can set *e.g.* $P_0 = 0.2$ as follows:

```
SSOpts['SubsetSim'] = {'p0': 0.2}
```

- **Specify the number of samples in each subset:** the number of samples in each subset N_S can be specified by using the `['Simulation']['BatchSize']` key . To set it to $N_S = 1000$ one can write:

```
SSOpts['Simulation'] = {'BatchSize': 1000}
```

For a comprehensive overview of the available options specific to subset simulation see Table 9, page 47.

2.3.6 Adaptive Kriging Monte Carlo Simulation (AK-MCS)

Adaptive Kriging Monte Carlo simulation method with default values (see Table 11 and the related linked tables for details on the defaults) can be deployed in UQ[PY]LAB with the following code:

```
AKOpts = {
    "Type": "Reliability",
    "Method":"AKMCS"
}
AKAnalysis = uq.createAnalysis(AKOpts)
```

Once the analysis is complete, a report with the AK-MCS results can be printed on screen by:

```
uq.print(AKAnalysis)
```

which produces the following:

```
--------------------------------------------------
AK-MCS
--------------------------------------------------
Pf                1.4200e-03
Beta              2.9845
CoV               0.0839
ModelEvaluations  17
PfCI              [1.1866e-03 1.6534e-03]
BetaCI            [2.9377e+00 3.0391e+00]
```

```
PfMinus/Plus      [1.4200e-03 1.4200e-03]
-------------------------------------------------
-----------------------------------------
```

The results can be visualized graphically as follows:

```
uq.display(AKAnalysis)
```

which produces the images in Figure 10. Note that the plot on the right of Figure 10 is only available when the input is two-dimensional. Additionally, if the verbosity is set to `['Display']` ≥ 5 the AK-MCS analysis will plot the convergence of P_f and β while the analysis is running.

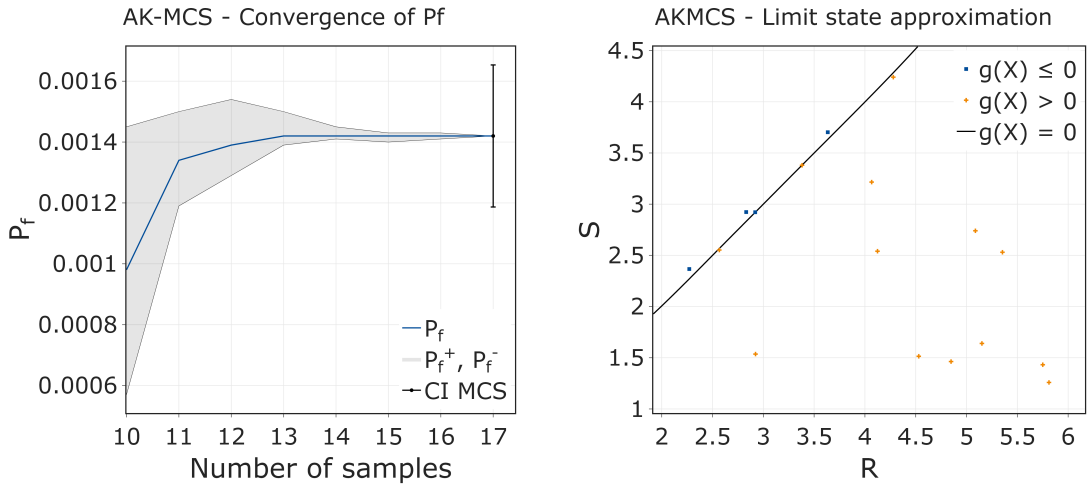

Figure 10: Graphical visualization of the convergence of the AK-MCS analysis in Section 2.3.6.

Note: In the preceding example no specifications are provided. If not further specified the Monte Carlo simulation runs with the following defaults:

- Confidence level: 0.05;
- Maximum number of evaluated samples: 10^5;
- Number of samples evaluated per batch: 10^4;
- Failure is defined for: limit-state $g(x) \leq 0$;
- Type of metamodel: `'Kriging'`;
- Learning function to determine the best next sample(s): `'U'`;
- Convergence criterion for the adaptive ED algorithm: `'stopU'`;
- Number of samples added to the ED for the metamodel: 10^3;
- Number of samples in the initial ED: $N_{ini} = \max(10, 2M)$
- Initial ED sampling strategy: `'LHS'`.

2.3.6.1 Accessing the results

The results from the AK-MCS algorithm are stored in the `AKAnalysis['Results']` dictionary:

```
{
    'Pf': 0.00142,
    'Beta': 2.984545456825952,
    'CoV': 0.08385853278663276,
    'ModelEvaluations': 17,
    'PfCI': [0.0011866092202373964, 0.0016533907797626035],
    'BetaCI': [2.9376799928499784, 3.0390545929919957],
    'Kriging': 'Model 2',
    'History': {...}
}
```

The keys in the `Results` dictionary have the same meaning as their counterparts in Monte Carlo simulation. Further, the key `Kriging` contains a string with a UQ[PY]LAB name of the Kriging metamodel. To retrieve the metamodel, one can write:

```
parentName = AKMCSAnalysis['Name']
objPath    = 'Results.Kriging'
myKriging  = uq.extractFromAnalysis(parentName=parentName,objPath=objPath)
```

This metamodel can be reused within UQ[PY]LAB for any other purpose, see UQ[PY]LAB User Manual – Kriging (Gaussian process modelling) for details.

2.3.6.2 Advanced options

AK-MCS uses the same advanced options as Monte Carlo simulation described in Section 2.3.3.2, as well as some additional options. The most important are summarized in the following:

- **Convergence criterion**: There are three different convergence criteria mentioned in Section 1.6.1.2. They are all available and can be specified *e.g.*criterion on failure probability:

  ```
  AKMCSOpts['AKMCS'] = {'Convergence': 'stopPf'}
  ```

- **Specify the Kriging metamodel**: The specifications for the Kriging metamodel (see also UQ[PY]LAB User Manual – Kriging (Gaussian process modelling)) can be set in the key `['AKMCS']['Kriging']` *e.g.*for an ordinary Kriging model:

  ```
  AKMCSOpts['AKMCS'] = {
      'Kriging': {
          'Trend': {
              'Type': 'ordinary'
          }
      }
  }
  ```

- **Specify the initial experimental design**: Apart from specifying a number of points and a sampling strategy, the initial experimental design can be specified by providing

$\mathcal{X} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N_0)}\}$ in the matrix X and the corresponding limit-state function values $\{g(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N_0)}\}$ in G:

```
AKOpts['AKMCS'] = {
    'IExpDesign': {
        'X': X.tolist(),
        'G': G.tolist()
    }
}
```

- **Specify the number of added experimental design points**: The maximum number of samples added to the experimental design of the Kriging metamodel can be specified to *e.g.*100:

  ```
  AKOpts['AKMCS'] = {'MaxAddedED': 100}
  ```

  Note that the total number of runs of the limit-state function then is at most the initial ED size plus the above number.

- **Use a PC-Kriging metamodel**: Instead of Kriging, a PC-Kriging model (see also UQ[PY]LAB User Manual – PC-Kriging) can be used as a surrogate model in AK-MCS.

  ```
  AKOpts['AKMCS'] = {'MetaModel': 'PCK'}
  ```

  Specific options of the PCK model can be added in the key `['AKMCS']['PCK']`. As an example, a Gaussian correlation function in PC-Kriging is set as:

  ```
  AKMCSOpts['AKMCS'] = {
      'PCK': {
          'Kriging': {
              'Corr': {
                  'Family': 'Gaussian'
              }
          }
      }
  }
  ```

For an overview of the advanced options available for the AK-MCS method, refer to Table 11, page 48.

2.4 Advanced limit-state function options

2.4.1 Specify failure threshold and failure criterion

While it is normally good practice to define the limit-state function directly as a UQ[PY]LAB MODEL object as in Section 2.2, in some cases it can be useful to be able to create one from small modifications of existing MODEL objects. A typical scenario where this is apparent is when the same objective function needs to be tested against a set of different failure thresholds, *e.g.* for a parametric study. In this case, the limit-state specifications can be modified. As an example, when $g(\boldsymbol{x}) \leq T = 5$ defines the failure criterion, one can use the following syntax:

```
MCOpts['LimitState'] = {
    'Threshold': 5,
    'CompOp': '<='
}
```

UQ[PY]LAB offers several possibilities to create simple (or arbitrarily complex) objective functions from existing MODEL objects (see also UQ[PY]LAB User Manual – the INPUT module).
For an overview of the advanced options for the limit-state function, refer to Table 4, page 44.

2.4.2 Vector Output

In case the limit-state function $g(\boldsymbol{X})$ results in a vector rather than a scalar value, the structural reliability module estimates the failure probability for each component independently.

Note: There is no system-type reasoning implemented to combine the failure probabilities of each component.

However, the implemented methods make use of evaluations of the limit-state function if available, as follows:

- **Monte Carlo simulation**: The enrichment of the sample size is increased until the convergence criteria are fulfilled for *all* components.
- **Subset Simulation**: The first batch of samples (MCS) is reused for every output component limit-state.
- **AK-MCS**: The initial experimental design for the Kriging model of output component i consists of the final experimental design of component $i-1$.

2.5 Excluding parameters from the analysis

In various usage scenarios (*e.g.* parametric studies) one or more input variables may be set to fixed constant values. This can have important consequences for many of the methods available in UQ[PY]LAB *e.g.* FORM/SORM and AK-MCS, whose costs increase significantly with the number of input variables. Whenever applicable, UQ[PY]LAB will appropriately account for the set of constant input parameters and exclude them from the analysis so as to avoid unnecessary costs. This process is transparent to the user as the analysis results will still show the excluded variables, but they will not be included in the calculations.
To set a parameter to constant, the following command can be used when the probabilistic input is defined (See UQ[PY]LAB User Manual – the INPUT module):

```
InputOpts['Marginals']['Type'] = 'Constant'
InputOpts['Marginals']['Parameters'] = [value]
```

Furthermore, when the standard deviation of a parameter equals zero, UQ[PY]LAB treats it as a `Constant`. For example, the following uniformly distributed variable whose upper and lower bounds are identical is automatically set to a constant with value 1:

```
InputOpts['Marginals']['Type'] = 'Uniform'
InputOpts['Marginals']['Parameters'] = [1, 1]
```

Chapter 3

Reference List

How to read the reference list

Python dictionaries play an important role throughout the UQLAB syntax. They offer a natural way to semantically group configuration options and output quantities. Due to the complexity of the algorithms implemented, it is not uncommon to employ nested dictionaries to fine-tune the inputs and outputs. Throughout this reference guide, a table-based description of the configuration dictionaries is adopted.
The simplest case is given when a value of a dictionary key is a simple value or a list:

Table X: `Input`			
●	`Name`	String	A description of the field is put here

which corresponds to the following syntax:

```
Input = {
  'Name' : 'My Input'
}
```

The columns, from left to right, correspond to the name, the data type and a brief description of each key-value pair. At the beginning of each row a symbol is given to inform as to whether the corresponding key is mandatory, optional, mutually exclusive, etc. The comprehensive list of symbols is given in the following table:

●	Mandatory
□	Optional
⊕	Mandatory, mutually exclusive (only one of the keys can be set)
⊞	Optional, mutually exclusive (one of them can be set, if at least one of the group is set, otherwise none is necessary)

When the value of one of the keys of a dictionary is a dictionary itself, a link to a table that describes the structure of that nested dictionary is provided, as in the case of the `Options` key in the following example:

Table X: `Input`			
●	`Name`	String	Description
☐	`Options`	Table Y	Description of the `Options` dictionary

Table Y: `Input.Options`			
●	`Field1`	String	Description of `Field1`
☐	`Field2`	Double	Description of `Field2`

In some cases, an option value gives the possibility to define further options related to that value. The general syntax would be:

```
Input = {
  'Option1' : 'VALUE1',
  'VALUE1' : {
    'Val1Opt1' : ... ,
    'Val1Opt2' : ...
  }
}
```

This is illustrated as follows:

Table X: `Input`			
●	`Option1`	String	Short description
		`'VALUE1'`	Description of `'VALUE1'`
		`'VALUE2'`	Description of `'VALUE2'`
⊞	`VALUE1`	Table Y	Options for `'VALUE1'`
⊞	`VALUE2`	Table Z	Options for `'VALUE2'`

Table Y: `Input.VALUE1`			
☐	`Val1Opt1`	String	Description
☐	`Val1Opt2`	Float	Description

Table Z: `Input.VALUE2`			
☐	`Val2Opt1`	String	Description

☐	`Val2Opt2`	Float	Description

3.1 Create a reliability analysis

Syntax

```
myAnalysis  = uq.createAnalysis(ROpts)
```

Input

All the parameters required to determine the analysis are to be given as keys of the structure `ROpts`. Each method has its own options, that will be reviewed in different tables. The options described in Table 3 are common to all methods.

Table 3: `ROpts`

●	`Type`	`'uq_reliability'`	Identifier of the module. The options corresponding to other types are in the corresponding guides.
●	`Method`	String	Type of structural reliability method. The available options are listed below:
		`'MCS'`	Monte Carlo simulation.
		`'FORM'`,	First order reliability method.
		`'SORM'`,	Second order reliability method.
		`'IS'`	Importance sampling.
		`'Subset'`	Subset simulation.
		`'AKMCS'`	Adaptive Kriging Monte Carlo Simulation (AK-MCS).
☐	`Name`	String	Name of the module. If not set by the user, a unique string is automatically assigned to it.
☐	`Input`	INPUT object	INPUT object used in the analysis. If not specified, the currently selected one is used.
☐	`Model`	MODEL object	MODEL object used in the analysis. If not specified, the currently selected one is used.
☐	`LimitState`	See Table 4	Specification of the limit-state function.

☐	`Display`	String default: `'standard'`	Level of information displayed by the methods.
		`'quiet'`	Minimum display level, displays nothing or very few information.
		`'standard'`	Default display level, shows the most important information.
		`'verbose'`	Maximum display level, shows all the information on runtime, like updates on iterations, etc.
☐	`Simulation`	See Table 5	Options key for the simulation methods. Only applies when `ROpts['Method']` is `'MCS'`, `'IS'`, `'SS'`, or `'AKMCS'`.
☐	`FORM`	See Table 6	Options key for the FORM algorithm methods. Only applies when `ROpts['Method']` is `'FORM'`, `'SORM'`, or `'IS'`.
☐	`Gradient`	See Table 7	Options key for computing the gradient. It applies to the methods that use FORM, namely, when `ROpts['Method']` is `'FORM'`, `'SORM'`, or `'IS'`.
☐	`IS`	See Table 8	Options key for importance sampling. It applies only when `ROpts['Method']` is `'IS'`.
☐	`Subset`	See Table 9	Options key for subset simulation. It applies only when `ROpts['Method']` is `'Subset'`.
☐	`AKMCS`	See Table 11	Options key for the adaptive experimental design algorithm in AK-MCS. This applies when `ROpts['Method']` is `'AKMCS'`.
☐	`SaveEvaluations`	Logical default: `True`	Storage or not of performed evaluations of the limit-state function.
		`True`	Store the evaluations.
		`False`	Do not store the evaluations.

In order to perform a structural reliability analysis, the limit-state function $g(\boldsymbol{x})$ is compared

to a threshold value T (by default $T = 0$). In analogy with Eq. (1.1), failure is defined as $g(\boldsymbol{x}) \leq T$. Alternatively, failure can be specified as $g(\boldsymbol{x}) \geq T$ by adjustment of the key `ROpts['LimitState']['CompOp']` to `'>='`. The relevant options are summarized in Table 4:

Table 4: `ROpts['LimitState']`			
☐	`Threshold`	Float default: 0	Threshold T, compared to the limit-state function $g(\boldsymbol{x})$.
☐	`CompOp`	String default: `'<='`	Comparison operator for the limit-state function.
		`'<'`, `'<='`	Failure is defined by $g(\boldsymbol{x}) < T$.
		`'>'`, `'>='`	Failure is defined by $g(\boldsymbol{x}) > T$.

The available methods to perform structural reliability analysis are Monte Carlo simulation, importance sampling, subset simulation, AK-MCS, FORM, and SORM. The first four methods share the simulation options. FORM and SORM are gradient-based, so they allow the user to specify the finite difference options as well as the algorithm options.
In Table 5, the options for the simulation methods (Monte Carlo, importance sampling, subset simulation and AK-MCS) are shown:

Table 5: `ROpts['Simulation']`			
☐	`Alpha`	Float default: 0.05	Confidence level α. For the Monte Carlo estimators, a confidence interval is constructed with confidence level $1 - \alpha$.
☐	`MaxSampleSize`	Integer default: 10^3 for `'IS'`; 10^5 otherwise	Maximum number of samples to be evaluated. If there is no target coefficient of variation (CoV), this is the total number of samples to be evaluated. If the target CoV is present, the method will run until `TargetCoV` or `MaxSampleSize` is reached. In this case, the default value of `MaxSampleSize`, if not specified in the options, is `Inf`, *i.e.*the method will run until the target CoV is achieved.

☐	`TargetCoV`	Float	Target coefficient of variation. If present, the method will run until the estimate of the CoV (Eq. (1.40)) is below `TargetCoV` or until `MaxSampleSize` function evaluations are performed. The value of the coefficient of variation of the estimator is checked after each `BatchSize` evaluations. By default this option is disabled. Note: this option has no effect in `Method = 'Subset'` and `'AKMCS'`.
☐	`BatchSize`	Integer default: 10^4 for `'MCS'` and `'AKMCS'`; 10^3 for `'Subset'`; 10^2 for `'IS'`	Number of samples that will be evaluated at once. Note that this option has no effect in `Method = 'AKMCS'`.

Note: In order to use importance sampling after an already computed FORM analysis, one can provide these results to the analysis options in order to avoid repeating FORM. If `FORMResults` is a structure containing the results of a FORM analysis, the syntax reads:

```
ISOpts['Type'] = 'Reliability;
ISOpts['Method'] = 'IS'
ISOpts['FORM'] = FORMResults
ISAnalysis = uq.createAnalysis(ISOpts)
```

The FORM algorithm has special parameters that can be tuned in the key `FORM` of the options. These parameters also affect the methods that depend on FORM, namely importance sampling and SORM. These are listed in Table 6.

Table 6: `ROpts['FORM']`

☐	`Algorithm`	String default: `'iHLRF'`	Algorithm used to find the design point.
		`'iHLRF'`	Improved HLRF.
		`'HLRF'`	HLRF.
☐	`StartingPoint`	$1 \times M$ List with Float entries default: `np.zeros(M).tolist()`	Starting point for the Rackwitz-Fiessler algorithm.

☐	`StopU`	Float default: 10^{-4}	Tolerance value for the Rackwitz-Fiessler algorithm on the design point. The algorithm will stop when $\lvert \boldsymbol{U}_{k+1} - \boldsymbol{U}_k \rvert < StopU$.
☐	`StopG`	Float default: 10^{-6}	Tolerance value for the Rackwitz-Fiessler algorithm on the limit-state function value. The algorithm will stop when $\lvert \frac{G(\boldsymbol{U}_k)}{G(\boldsymbol{U}_0)} \rvert < StopG$.
☐	`MaxIterations`	Integer default: 100	Maximum number of iterations allowed in the Rackwitz-Fiessler algorithm. If this property should be ignored, it can be set to `Inf`.

Since FORM is a gradient-based method, the gradient of the limit-state function needs to be computed. This is done using finite differences. The options for the differentiation are listed in Table 7.

Table 7: `ROpts['Gradient']`

☐	`h`	Float default: 10^{-3}	Value of the difference for the scheme.
☐	`Method`	String default: `'forward'`	Specifies the type of finite differences scheme to be used.
		`'forward'`	Forward finite differences. $\frac{\partial g}{\partial x_i}$ is approximated using $g(\boldsymbol{x})$ and $g(\boldsymbol{x} + h\boldsymbol{e}_i)$.
		`'backward'`	$\frac{\partial g}{\partial x_i}$ is approximated using $g(\boldsymbol{x})$ and $g(\boldsymbol{x} - h\boldsymbol{e}_i)$.
		`'centered'`	$\frac{\partial g}{\partial x_i}$ is approximated using $g(\boldsymbol{x} + h\boldsymbol{e}_i)$ and $g(\boldsymbol{x} - h\boldsymbol{e}_i)$. (More accurate and more costly.)

The options specifically set for the importance sampling are presented in Table 8. Note that the options of `Simulation` and `FORM` are also processed in the case of importance sampling due to the nature of the MCS, FORM and IS.

Table 8: `ROpts['IS']`

☐	`Instrumental`	$1 \times N_{out}$ INPUT object or Dictionary	Instrumental distribution defined as either a dictionary of input marginals and copula or an INPUT object (refer to UQ[PY]LAB User Manual – the INPUT module for details).
☐	`FORM`	FORM ANALYSIS object or `FORMAnalysis['Results']` Dictionary	FORM results computed previously. See Section 2.3.4.2 for details.

The options specifically set for subset simulation are presented in Table 9. Note that the options of `Simulation` are also processed in the case of subset simulation due to the similar nature of Monte Carlo simulation and subset simulation.

Table 9: `ROpts['Subset']`

☐	`p0`	Float default: 0.1	Target conditional failure probability of auxiliary limit-states ($0 <$ `p0` ≤ 0.5).
☐	`Proposal`	See Table 10	Description of the proposal distribution in the Markov Chain.
☐	`MaxSubsets`	Integer default: $\frac{\texttt{MaxSampleSize}}{\texttt{BatchSize}\cdot(1-\texttt{p0})}$	Maximum number of subsets. In the subset simulation algorithm, the maximum number of subsets is set to the minimum of `MaxSubsets` and $\frac{\texttt{MaxSampleSize}}{\texttt{BatchSize}\cdot(1-\texttt{p0})}$.

The settings of the Markov Chain Monte Carlo simulation in subset simulation are summarized in Table 10. Note that the default values are taken from Au and Beck (2001).

Table 10: `ROpts['Subset']['Proposal']` (Proposal distributions)

☐	`Type`	String default: `'Uniform'`	Type of proposal distribution (in the standard normal space).
		`'Gaussian'`	Gaussian distribution.
		`'Uniform'`	Uniform distribution.

☐	`Parameters`	Float default: 1	Parameter of the proposal distribution. Corresponds to the standard deviation for a Gaussian distribution and the half-width for the uniform distribution.

AK-MCS is a combination of Kriging metamodels and Monte Carlo simulation. The options for the Kriging metamodel and the adaptive experimental design algorithm are listed here.

Table 11: `ROpts['AKMCS']`

☐	`MetaModel`	String default: `'Kriging'`	Choice of metamodel in AK-MCS.
⊞	`Kriging`	Dictionary	Kriging options when key `MetaModel = 'Kriging'`. If none is set, then the default Kriging options are used (refer to UQ[PY]LAB User Manual – Kriging (Gaussian process modelling)). Note that a small nugget of 10^{-10} is added by default to the correlation options to improve numerical stability.
⊞	`PCK`	Dictionary	PC-Kriging options when key `MetaModel = 'PCK'`. If none is set, then the default PC-Kriging options are used (refer to UQ[PY]LAB User Manual – PC-Kriging). Note that a small nugget of 10^{-10} is added by default to the correlation options to improve numerical stability.
☐	`LearningFunction`	String default: `'U'`	Learning function to determine the best next sample(s) to be added to the experimental design.
		`'U'`	U-function (see Eq. (1.59)).
		`'EFF'`	Expected feasibility function (see Eq. (1.61)).
☐	`Convergence`	String default: `'stopU'`	Convergence criterion for the adaptive experimental design algorithm.
		`'stopU'`	Convergence when $\min U(\boldsymbol{x}) \geq 2$ (see Echard et al. (2011)) on the candidate set.

		`'stopPf'`	Convergence criterion based on the convergence of the failure probability estimate (see Eq. (1.63)).
		`'stopBeta'`	Convergence criterion based on the convergence of the reliability index estimate (see Eq. (1.66)).
☐	`MaxAddedED`	Integer default: 1000	Number of samples added to the experimental design of the Kriging metamodel.
☐	`IExpDesign`	See Table 12	Specification of the initial experimental design of the metamodel.

The initial experimental design of AK-MCS can either be given by a number of samples and a sampling method or by a matrix containing the set of input samples and the corresponding values of the limit-state function.

Table 12: `ROpts['AKMCS']['IExpDesign']`

☐	`N`	Integer default: 10	Number of samples in the initial experimental design.
☐	`Sampling`	String default: `'LHS'`	Sampling techniques of the initial experimental design. See UQ[PY]LAB User Manual – the INPUT module for more sampling techniques.
☐	`X`	$N \times M$ List of lists with Float entries	Matrix containing the initial experimental design.
☐	`G`	$N \times N_{out}$ List of lists with Float entries	Vector containing the responses of the limit-state function corresponding to the initial experimental design, corrected by the threshold k (see also Table 4): $(g(\boldsymbol{x}) - T)$ for `Criterion = '<='`, $(T - g(\boldsymbol{x}))$ for `Criterion = '>='`.

3.2 Accessing the results

Syntax

```
myAnalysis  = uq.createAnalysis(ROpts)
```

Output

The information stored in the `myAnalysis['Results']` dictionary depends on which kind of analysis is performed. In the sequel, the results for each of the methods are reviewed.

- Monte Carlo - Table 13
- FORM - Table 15
- SORM - Table 17
- Importance sampling - Table 18
- Subset simulation - Table 19
- AK-MCS - Table 21

3.2.1 Monte Carlo

The results are summarized in Table 13.

Table 13: `myAnalysis['Results']`

`Pf`	Float	Estimator of the failure probability, $P_{f,\text{MCS}}$.
`Beta`	Float	Associated reliability index, $\beta_{\text{MCS}} = -\Phi^{-1}(P_{f,\text{MCS}})$.
`CoV`	Float	Coefficient of variation.
`ModelEvaluations`	Integer	Total number of model evaluations performed during the analysis.
`PfCI`	1×2 List with Float entries	Confidence interval of the failure probability.
`BetaCI`	1×2 List with Float entries	Confidence interval of the associated reliability index.
`History`	See Table 14	If the simulation is carried out using batches of points, `History[i-1]` contains results obtained after the i-th batch.

If the simulation has been carried out by using various batches of points, the information on the convergence in each step is stored in the structure `History`. Its contents are described in Table 14.

Table 14: `myAnalysis['Results']['History']`

`Pf`	Float	Failure probability estimate after each batch.
`CoV`	Float	Coefficient of variation after each batch.
`Conf`	Float	Confidence interval after each batch.
`X`	$N \times M$ List of lists with Float entries	Matrix containing the input vectors in the original space evaluated by the limit-state function.
`U`	$N \times M$ List of lists with Float entries	Matrix containing the input vectors in the standard normal space evaluated by the limit-state function.
`G`	$N \times N_{out}$ List of lists with Float entries	Values of the limit-state function along the FORM iterations *i.e.* $g(\boldsymbol{x}_k) - T$ for `Criterion = '<='`, $T - g(\boldsymbol{x}_k)$ for `Criterion = '>='`.

3.2.2 FORM and SORM

FORM and SORM methods are very close in terms of calculations. Indeed, SORM can be understood as a correction of the FORM estimation of the probability. Therefore, the results dictionaries are very similar. The results of FORM are shown in Table 15. When executing SORM, some keys will be added to the FORM `Results` dictionary, shown in Table 17.

Table 15: `myAnalysis['Results']`

`BetaHL`	Float	Hasofer-Lind reliability index, β_{HL}.
`Pf`	Float	Estimator of the failure probability, $P_{f,\text{FORM}} = \Phi(-\beta_{HL})$.
`ModelEvaluations`	Integer	Total number of model evaluations performed to solve the analysis.
`Ustar`	$1 \times M$ List with Float entries	Design point $\boldsymbol{U}^*$ in the standard normal space.
`Xstar`	$1 \times M$ List with Float entries	Design point $\boldsymbol{X}^*$ the original space.
`Importance`	$1 \times M$ List with Float entries	Importance factors S_i.
`Iterations`	Integer	Number of iterations carried out by the optimization algorithm.

`Method`	String	Method used to solve the analysis.
`History`	See Table 16	Dictionary with information about the algorithm steps and runtime information.

The key `History` of the results contains more detailed information extracted from the algorithm steps.

Table 16: `myAnalysis['Results']['History']`

`ExitFlag`	String	Reason why the algorithm stopped.
`BetaHL`	Float	Values of the reliability index along the FORM iterations.
`OriginValue`	Float	Value of the limit-state function $G(\boldsymbol{u} = \mathbf{0})$.
`G`	$N \times N_{out}$ List of lists with Float entries	Values of the limit-state function along the FORM iterations *i.e.* $g(\boldsymbol{x}_k) - T$ for `Criterion = '<='`, $T - g(\boldsymbol{x}_k)$ for `Criterion = '>='`.
`Gradient`	Float	Values of the gradient of the limit-state function along the FORM iterations.
`StopU`	Float	Values of the stopping criterion on the design point convergence, $\lvert\boldsymbol{U}_{k+1} - \boldsymbol{U}_k\rvert$, along the FORM iterations.
`StopG`	Float	Values of the stopping criteria on the limit-state function, $\lvert\frac{G(\boldsymbol{U}_k)}{G(\boldsymbol{U}_0)}\rvert$, along the FORM iterations.
`U`	$N \times M$ List of lists with Float entries	Coordinates of the points $\boldsymbol{U}_k$ in the standard normal space.
`X`	$N \times M$ List of lists with Float entries	Coordinates of the points $\mathbf{X}_k$ in the original space.

If SORM is also performed, two keys are added to the `Results`, and two keys are added to `Results['History']`, as shown in Table 17.

Table 17: `myAnalysis['Results']`

`PfSORM`	Float	SORM estimator of the failure probability, $P_{f,\text{SORM}}$, using Hohenbichler's formula.
`PfSORMBreitung`	Float	SORM estimator of the failure probability, $P_{f,\text{SORM}}$, using Breitung's formula.
`BetaSORM`	Float	Associated reliability index $\beta_{\text{SORM}} = -\Phi^{-1}(P_{f,\text{SORM}})$, using Hohenbichler's formula.
`BetaSORMBreitung`	Float	Associated reliability index $\beta_{\text{SORM}} = -\Phi^{-1}(P_{f,\text{SORM}})$, using Breitung's formula.
`History['FORMEvals']`	Integer	Number of model evaluations carried out to perform the FORM analysis.
`History['Hessian']`	$M \times M$ List of lists with Float entries	Hessian matrix of the limit-state function at the design point, $\boldsymbol{U}^*$.

3.2.3 Importance sampling

Since importance sampling is a simulation method, the dictionary of the results is similar to the one of Monte Carlo simulation. The results are listed in Table 18.

Table 18: `myAnalysis['Results']`

`Pf`	Float	Estimator of the failure probability, $P_{f,\text{IS}}$.
`Beta`	Float	Associated reliability index, $\beta_{\text{IS}} = -\Phi^{-1}(P_{f,\text{IS}})$.
`CoV`	Float	Coefficient of variation.
`ModelEvaluations`	Integer	Total number of model evaluations performed during the analysis.
`PfCI`	1×2 List with Float entries	Confidence interval of the failure probability.
`BetaCI`	1×2 List with Float entries	Confidence interval of the associated reliability index.
`FORM`	See Table 15	Results of the FORM analysis used to find the design point.

`History`	See Table 14	If importance sampling is carried out using batches of points, `History[i-1]` contains results obtained after the i-th batch.

3.2.4 Subset simulation

Since subset simulation is a simulation method, the dictionary of the results is similar to the one of Monte Carlo simulation. The results are listed in Table 19 and Table 20.

Table 19: `myAnalysis['Results']`

`Pf`	Float	Estimator of the failure probability, $P_{f,\mathrm{SS}}$.
`Beta`	Float	Associated reliability index, $\beta_{\mathrm{SS}} = -\Phi^{-1}(P_{f,\mathrm{SS}})$.
`CoV`	Float	Coefficient of variation.
`ModelEvaluations`	Integer	Number of model evaluations during the analysis.
`PfCI`	1×2 List with Float entries	Confidence interval of the failure probability.
`BetaCI`	1×2 List with Float entries	Confidence interval of the associated reliability index.
`NumberSubsets`	Integer	Number of auxiliary subsets during the analysis.
`History`	See Table 20	Data related to each subset.

The key `History` of the results contains more detailed information extracted from the algorithm steps.

Table 20: `myAnalysis['Results']['History']`

`delta2`	Float	δ_j^2 for estimating the coefficient of variation (see Eq. (1.55)).
`q`	Float	Intermediate limit-state thresholds.
`X`	List	Samples in the input space of each subset.
`U`	List	Samples in the standard normal space of each subset.

`G`	List	Values of the limit-state function of each sample in each subset corrected by the threshold T (see also Table 4): $(g(\boldsymbol{x}) - T)$ for `Criterion = '<='`, $(T - g(\boldsymbol{x}))$ for `Criterion = '>='`.
`Pfcond`	Float	Conditional failure probability estimates $\mathbb{P}(\mathcal{D}_{i+1}\|\mathcal{D}_i)$.
`gamma`	Float	γ_j for computing the coefficient of variation (see Eq. (1.56)).

3.2.5 AK-MCS

Since AK-MCS relies upon Monte Carlo simulation, the dictionary of the results is similar to the one of Monte Carlo simulation. The results are listed in Table 21 and Table 22.

Table 21: `myAnalysis['Results']`

`Pf`	Float	Estimator of the failure probability, $P_{f,\text{AK-MCS}}$.
`Beta`	Float	Associated reliability index, $\beta_{\text{AK-MCS}} = -\Phi^{-1}(P_{f,\text{AK-MCS}})$.
`CoV`	Float	Coefficient of variation.
`ModelEvaluations`	Integer	Total number of model evaluations performed during the analysis.
`PfCI`	1×2 List with Float entries	Confidence interval of the failure probability.
`BetaCI`	1×2 List with Float entries	Confidence interval of the associated reliability index.
`Kriging`	String	Name of the final Kriging metamodel.
`PCK`	String	Name of the final PC-Kriging metamodel.
`History`	See Table 22	Contains intermediate results along the AK-MCS iterations.

Table 22: `myAnalysis['Results']['History']`

`Pf`	Float	History of the estimate failure probability.
`PfLower`	Float	History of the estimated lower bound of the failure probability P_f^-.
`PfUpper`	Float	History of the estimated upper bound of the failure probability P_f^+.
`NSamples`	Integer	Number of samples added to the experimental design at each iteration.
`NInit`	Integer	Number of samples in the initial experimental design.
`X`	List	Samples in the input space of each subset.
`G`	List	Values of the limit-state function of each sample in each subset corrected by the threshold T (see also Table 4): $(g(\boldsymbol{x}) - T)$ for `Criterion = '<='`, $(T - g(\boldsymbol{x}))$ for `Criterion = '>='`.

References

Au, S. K. and J. L. Beck (2001). Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic Engineering Mechanics 16*(4), 263–277. 13, 14, 32, 47

Bichon, B. J., M. S. Eldred, L. Swiler, S. Mahadevan, and J. McFarland (2008). Efficient global reliability analysis for nonlinear implicit performance functions. *AIAA Journal 46*(10), 2459–2468. 16

Breitung, K. (1989). Asymptotic approximations for probability integrals. *Probabilistic Engineering Mechanics 4*(4), 187–190. 9, 10

Cai, G. Q. and I. Elishakoff (1994). Refined second-order reliability analysis. *Struct. Saf. 14*(4), 267–276. 9, 10

Dani, V., T. P. Hayes, and S. M. Kakade (2008). Stochastic linear optimization under bandit feedback. In *The 21st Annual Conference on Learning Theory (COLT 2008)*. 16

Ditlevsen, O. and H. Madsen (1996). *Structural reliability methods*. J. Wiley and Sons, Chichester. 3

Dubourg, V. (2011). *Adaptive surrogate models for reliability analysis and reliability-based design optimization*. Ph. D. thesis, Université Blaise Pascal, Clermont-Ferrand, France. 16

Echard, B., N. Gayton, and M. Lemaire (2011). AK-MCS: an active learning reliability method combining Kriging and Monte Carlo simulation. *Struct. Saf. 33*(2), 145–154. 15, 16, 17, 18, 48

Ginsbourger, D., B. Rosspopoff, G. Pirot, N. Durrande, and R. Renard (2013). Distance-based Kriging relying on proxy simulation for inverse conditioning. *Advances in Water Resources 52*, 275–291. 16

Hasofer, A.-M. and N.-C. Lind (1974). Exact and invariant second moment code format. *J. Eng. Mech. 100*(1), 111–121. 6

Hohenbichler, M., S. Gollwitzer, W. Kruse, and R. Rackwitz (1987). New light on first- and second order reliability methods. *Struct. Saf. 4*, 267–284. 10

Lemaire, M. (2009). *Structural reliability*. Wiley. 3

Melchers, R.-E. (1999). *Structural reliability analysis and prediction*. John Wiley & Sons. 3, 12

Papaioannou, I., W. Betz, K. Zwirglmaier, and D. Straub (2015). MCMC algorithms for subset simulation. *Probabilistic Engineering Mechanics 41*, 89–103. 14

Rackwitz, R. and B. Fiessler (1978). Structural reliability under combined load sequences. *Computers & Structures 9*, 489–494. 7

Rubinstein, R.-Y. (1981). *Simulation and the Monte Carlo methods*. John Wiley & Sons. 11

Schöbi, R., B. Sudret, and S. Marelli (2016). Rare event estimation using Polynomial-Chaos-Kriging. *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part A: Civil Engineering*, D4016002. 15, 17, 18

Srinivas, N., A. Krause, S. Kakade, and M. Seeger (2012). Information-theoretic regret bounds for Gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory 58*(5), 3250–3265. 16

Sudret, B. (2007). Uncertainty propagation and sensitivity analysis in mechanical models - Contributions to structural reliability and stochastic spectral methods. Habilitation thesis, Université Blaise Pascal, Clermont-Ferrand, France. 1

Tvedt, L. (1990). Distribution of quadratic forms in normal space – Applications to structural reliability. *J. Eng. Mech. 116*(6), 1183–1197. 10

Zhang, Y. and A. Der Kiureghian (1995). Two improved algorithms for reliability analysis. In R. Rackwitz, G. Augusti, and A. Bori (Eds.), *Proc. 6th IFIP WG7.5 on Reliability and Optimization of Structural systems, Assisi, Italy*. Chapman & Hall, London. 8