ETH zürich

UQLab 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 UQLab

S. Marelli, and B. Sudret, UQLab: A framework for uncertainty quantification in Matlab, Proc. 2nd Int. Conf. on Vulnerability, Risk Analysis and Management (ICVRAM2014), Liverpool, United Kingdom, 2014, 2554-2563.

How to cite this manual

S. Marelli, R. Schöbi, B. Sudret, UQLab user manual – Structural reliability (Rare event estimation), Report UQLab-V2.1-107, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland, 2024

BibTeX entry

```
@TechReport{UQdoc_21_107,
author = {Marelli, S. and Sch\"obi, R. and Sudret, B.},
title = {{UQLab user manual -- Structural reliability (Rare event estimation)}},
institution = {Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich,
Switzerland},
year = {2024},
note = {Report UQLab-V2.1-107}
}
```

List of contributors:

L. Fritsch	**Implementation of line sampling methods**
P.-R. Wagner	**Implementation of SSER**
C. Lamas	**Original implementation of FORM, SORM, Monte Carlo and importance sampling methods**

Document Data Sheet

Document Ref.	UQLAB-V2.1-107
Title:	UQLAB 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:	15/04/2024

Doc. Version	Date	Comments
V2.1	15/04/2024	UQLAB V2.1 release
V2.0	01/02/2022	UQLAB V2.0 release • added SSER method
V1.4	01/02/2021	UQLAB V1.4 release
V1.3	19/09/2019	UQLAB V1.3 release
V1.2	22/02/2019	UQLAB V1.2 release
V1.1	05/07/2018	UQLAB V1.1 release • Updated usage and reference list for AK-MCS
V1.0	01/05/2017	UQLAB V1.0 release • added PC-Kriging to AK-MCS • bugfixes and display improvements
V0.92	15/02/2016	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 UQLAB 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, Line Sampling) and state-of-the-art algorithms (AK-MCS, SSER) are available and can be easily deployed in association with other UQLAB 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 UQLAB structural reliability module.

Keywords: Structural Reliability, FORM, SORM, Importance Sampling, Monte Carlo Simulation, Subset Simulation, Line Sampling, AK-MCS, SSER, UQLAB, 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 UQLAB. Consistently with the overall design philosophy of UQLAB, 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

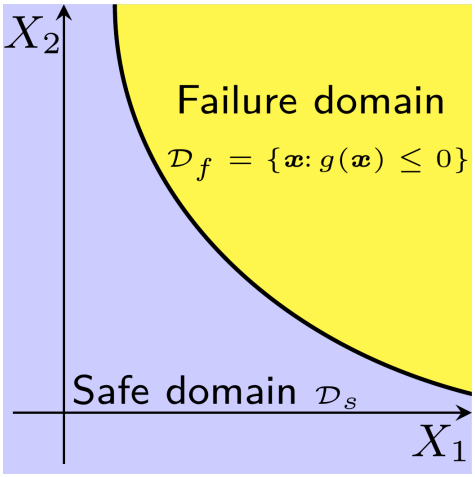

Figure 1: Schematic representation of the safe and failure domains $\mathcal{D}_s$ and $\mathcal{D}_f$ and the corresponding limit-state surface $g(\boldsymbol{x}) = 0$.

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 UQLAB:

- *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 UQLAB:

- *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.

- *Line Sampling* – it is based on drawing lines parallel to the important direction and estimating P_f based on the distance to the limit-state surface which using an appropriate root finder.

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.* UQLAB User Manual – Polynomial Chaos Expansionsand UQLAB 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 UQLAB, 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.

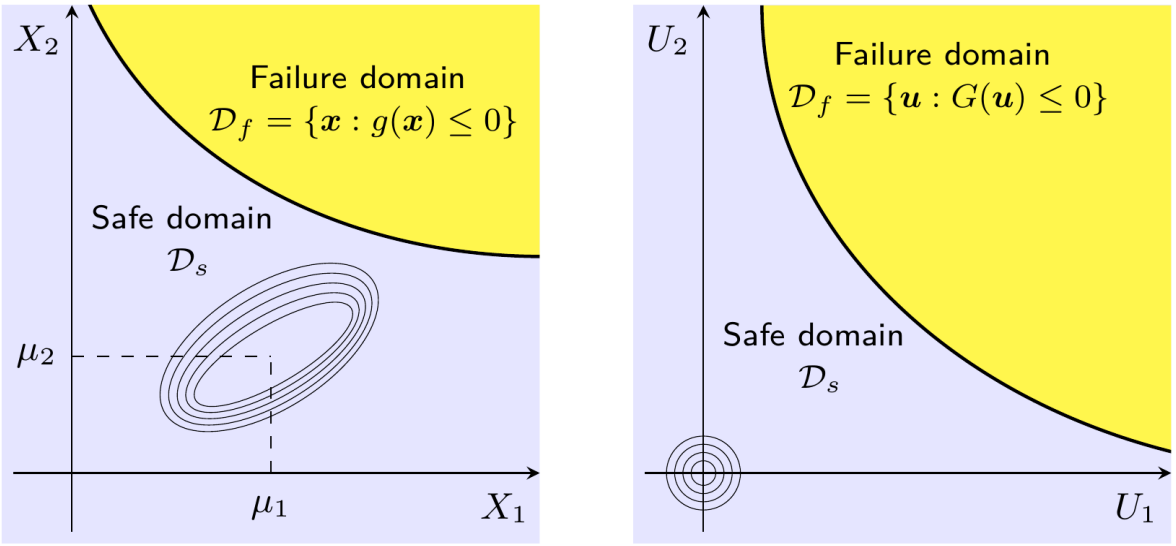

Figure 2: Graphical representation of the isoprobabilistic transform from physical to standard normal space in Eq. (1.5). From Sudret, 2015: Lectures on structural reliability and risk analysis.

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:

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

For details about the available isoprobabilistic transforms in UQLAB, please refer to the UQLAB User Manual – the INPUT module (Section 1.5).

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 de-

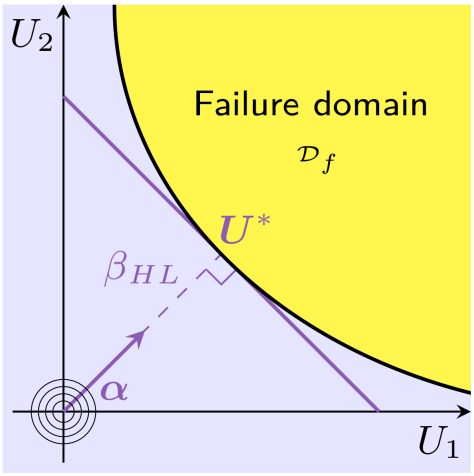

Figure 3: Graphical representation of the linearization of the limit-state function around the design point at the basis of the FORM estimation of P_f. From Sudret, 2015: Lectures on structural reliability and risk analysis.

cays 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.

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:

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

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

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

Note: In UQLAB 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

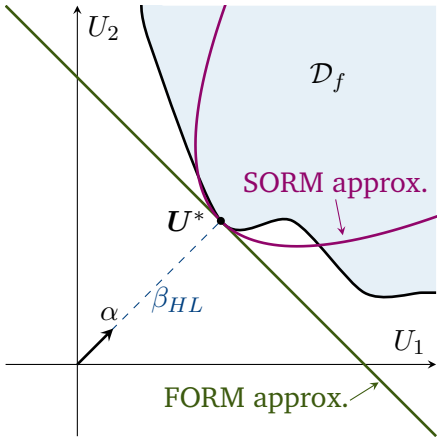

Figure 4: Comparison between FORM and SORM approximations of the failure domain for a simple 2-dimensional case.

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 $\boldsymbol{V}^* = \{0, \cdots, \beta\}^{\mathsf{T}}$ 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 $\boldsymbol{\alpha}$:

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

For small curvatures $\kappa_i \ll 1$, the failure probability P_f can be approximated by the Breitung formula (Breitung, 1984):

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

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 (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, the central limit theorem guarantees that $\widehat{P}_f$ follows asymptotically a normal distribution:

$$\widehat{P}_f \sim \mathcal{N}\left(P_f, \sigma_{\widehat{P}_f}\right), \tag{1.37}$$

where $\sigma_{\widehat{P}_f} = \sigma^2_{\mathbf{1}_{\mathcal{D}_f}}/N = P_f(1-P_f)/N$. By approximating the latter with $\widehat{\sigma}^2_{\widehat{P}_f} = \widehat{P}_f(1-\widehat{P}_f)/N$, we obtain the confidence interval for P_f as follows (Rubinstein, 1981):

$$P_f \in \left[\widehat{P}_f^- \overset{\text{def}}{=} \widehat{P}_f + \widehat{\sigma}_{P_f}\Phi^{-1}(\alpha/2),\ \widehat{P}_f^+ \overset{\text{def}}{=} \widehat{P}_f + \widehat{\sigma}_{P_f}\Phi^{-1}(1-\alpha/2)\right], \tag{1.38}$$

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:

$$CoV = \frac{\sigma_{\widehat{P}_f}}{\widehat{P}_f} = \sqrt{\frac{1-\widehat{P}_f}{N\widehat{P}_f}}. \tag{1.39}$$

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:

$$\beta_{\text{MCS}} = -\Phi^{-1}(\widehat{P}_f). \tag{1.40}$$

In analogy, upper and lower confidence bounds on β_{MCS} can be directly inferred from the

confidence bounds on $\widehat{P}_f$ in Eq. (1.38):

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

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:

$$P_f = \int_{\mathcal{D}_{\boldsymbol{X}}} \mathbf{1}_{D_f}(\boldsymbol{x}) \frac{f_{\boldsymbol{X}}(\boldsymbol{x})}{\Psi(\boldsymbol{x})} \Psi(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{\Psi}\left[\mathbf{1}_{\mathcal{D}_f}(\boldsymbol{X}) \frac{f_{\boldsymbol{X}}(\boldsymbol{X})}{\Psi(\boldsymbol{X})}\right], \tag{1.42}$$

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:

$$P_{f,\text{IS}} = \frac{1}{N} \sum_{k=1}^{N} \mathbf{1}_{\mathcal{D}_f}(\boldsymbol{x}^{(k)}) \frac{f_{\boldsymbol{X}}(\boldsymbol{x}^{(k)})}{\Psi(\boldsymbol{x}^{(k)})}. \tag{1.43}$$

In the standard normal space, Eq. (1.42) can be rewritten as:

$$P_f = \mathbb{E}_{\Psi}\left[\mathbf{1}_{\mathcal{D}_f}(\mathcal{T}^{-1}(\boldsymbol{U})) \frac{\varphi_M(\boldsymbol{U})}{\Psi(\boldsymbol{U})}\right]. \tag{1.44}$$

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):

$$\Psi(\boldsymbol{u}) = \varphi_M(\boldsymbol{u} - \boldsymbol{U}^*) \tag{1.45}$$

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:

$$P_{f,\text{IS}} = \frac{1}{N} \exp\left(-\beta^2_{HL}/2\right) \sum_{k=1}^{N} \mathbf{1}_{\mathcal{D}_f}\left(\mathcal{T}^{-1}(\boldsymbol{u}^{(k)})\right) \exp\left(-\boldsymbol{u}^{(k)} \cdot \boldsymbol{U}^*\right) \tag{1.46}$$

with corresponding variance:

$$\widehat{\sigma}^2_{P_{f,\text{IS}}} = \frac{1}{N} \frac{1}{N-1} \sum_{k=1}^{N} \left(\mathbf{1}_{\mathcal{D}_f}\left(\mathcal{T}^{-1}(\boldsymbol{u}^{(k)})\right) \frac{\varphi(\boldsymbol{u}^{(k)})}{\Psi(\boldsymbol{u}^{(k)})} - P_{f,\text{IS}}\right)^2. \tag{1.47}$$

The coefficient of variation and the confidence bounds $P^{\pm}_{f,\text{IS}}$ can be calculated analogously to Eqs. (1.39) and (1.38), 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:

$$\beta_{\text{IS}} = -\Phi^{-1}(\widehat{P}_{f,\text{IS}}), \tag{1.48}$$

with upper and lower bounds:

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

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. (1.39)). 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):

$$P_f = \mathbb{P}(\mathcal{D}_m) = P\left(\bigcap_{k=1}^{m} \mathbb{P}(\mathcal{D}_k)\right) = \mathbb{P}(\mathcal{D}_1) \prod_{i=1}^{m-1} \mathbb{P}(\mathcal{D}_{i+1}|\mathcal{D}_i)\,. \tag{1.50}$$

With an appropriate choice of the intermediate thresholds $t_1, \ldots, t_m$, Eq. (1.50) 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. (1.50) 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):

$$\mathbb{P}(\mathcal{D}_1) \approx \widehat{P}_1 = \frac{1}{N_S} \sum_{k=1}^{N_S} \mathbf{1}_{\mathcal{D}_1}(\boldsymbol{x}^{(k)}). \tag{1.51}$$

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 UQLAB 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. (1.50) with the current estimates of the conditional probabilities P_i:

$$\widehat{P}_{f,\mathrm{SS}} = \prod_{i=1}^{m} \widehat{P}_i = P_0^{m-1} \widehat{P}_m. \tag{1.52}$$

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. (1.52) is non-trivial. Au and Beck (2001) and Papaioannou et al. (2015) derived an estimate for the CoV of $\widehat{P}_f$:

$$CoV_f \approx \sum_{i=1}^{m} \delta_i^2, \tag{1.53}$$

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

$$\delta_i = \sqrt{\frac{1-P_i}{NP_i}\,(1+\gamma_i)}, \tag{1.54}$$

with

$$\gamma_i = 2\sum_{k=1}^{N/N_s}\left(1-\frac{kN_s}{N}\right)\rho_i(k), \tag{1.55}$$

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. (1.38). The corresponding generalized reliability index reads:

$$\beta_{\text{SS}} = -\Phi^{-1}(\widehat{P}_{f,\text{SS}}), \tag{1.56}$$

with upper and lower bounds:

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

1.5.4 Line Sampling

Line Sampling (LS) is a simulation method introduced by Koutsourelakis et al. (2004), offering an alternative to subset simulation and importance sampling for dealing with small failure probabilities. It requires fewer evaluations of the performance function compared to standard Monte Carlo simulation. Uniquely, LS utilizes lines, instead of points, to estimate failure probabilities. This method leverages critical properties of the standard normal space: independent marginals and orthogonality invariance. LS capitalizes on these properties by introducing a new coordinate system: $\boldsymbol{U}_{||}$, aligned with the important direction $\boldsymbol{\alpha}$, and $\boldsymbol{U}_{\perp}$, defined by the hyperplane orthogonal to $\boldsymbol{\alpha}$.

1.5.4.1 Standard Procedure

The LS procedure involves sampling points within an $(M-1)$-dimensional hyperplane $\mathcal{H}$ orthogonal to $\boldsymbol{\alpha}$. For each sampled point, a line parallel to $\boldsymbol{\alpha}$ is extended to intersect with the limit-state surface $G(\boldsymbol{U})=0$, as depicted in Figure 5. Assuming a semi-open failure domain, the failure probability along the i-th line is determined by

$$p^{(i)}_{f,\text{line}} = \Phi(-\beta^{(i)}). \tag{1.58}$$

Hence, identifying the intersection point—specifically, the distance $\beta^{(i)}$—is crucial. Various root-finding techniques, such as polynomial and spline interpolation, along with Newton's iterative method, are employed. The overall failure probability estimate is the mean of indi-

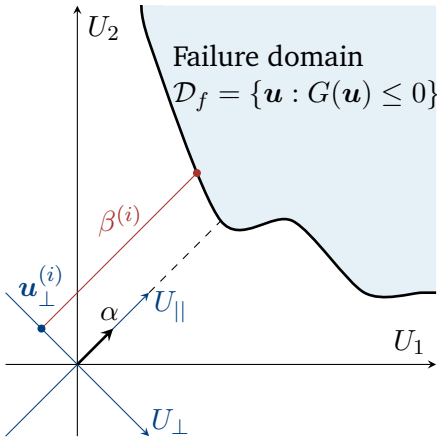

Figure 5: Line sampling procedure with new coordinates $\boldsymbol{U}_{||}$ and $\boldsymbol{U}_\perp$, sample on the hyperplane $\boldsymbol{u}_\perp^{(i)}$ and distance $\beta^{(i)}$.

vidual line failure probabilities:

$$P_{f,\mathsf{LS}} = \frac{1}{N}\sum_{i=1}^{N} p_{f,\mathsf{line}}^{(i)}. \tag{1.59}$$

The variance of this estimator is calculated from the variance of these probabilities:

$$\hat{\sigma}^2_{P_{f,\mathsf{LS}}} = \frac{1}{N}\left(\frac{1}{N-1}\sum_{i=1}^{N}\left(p_{f,\mathsf{line}}^{(i)} - P_{f,\mathsf{LS}}\right)^2\right). \tag{1.60}$$

Note that this procedure identifies only one critical direction, limiting its efficiency for complex failure domains.

1.5.4.2 Adaptive Procedure

An adaptive LS procedure, proposed by de Angelis et al. (2015), introduces two significant modifications:

1. Lines are processed in an optimal order: the next line is launched from the closest point sampled on the hyperplane $\boldsymbol{u}_\perp^{(i+1)}$ to the current line foot $\boldsymbol{u}_\perp^{(i)}$, thus leveraging the information on the previously processed lines.

2. The important direction $\boldsymbol{\alpha}$ is dynamically updated upon discovering a more probable failure point, allowing initial rough estimates of $\boldsymbol{\alpha}$, such as the normalized negative gradient at the origin.

This adaptive approach can reduce the variance of the failure probability estimate through the use of weighted estimators for the mean of the failure probability and the corresponding variance, as suggested by Papaioannou and Straub (2021). There, the estimator of P_f is derived as a weighted average of the probability estimates evaluated along each important

direction $\boldsymbol{\alpha}_k$. The estimator is given as:

$$P_{f,\mathsf{LS}} = \sum_{k=1}^{K} w_k \cdot p_{f,\mathsf{LS}}^{(k)}. \tag{1.61}$$

Here, K denotes the total number of important directions $\boldsymbol{\alpha}_k$, w_k is the weight corresponding to the k-the direction, and $p_{f,\mathsf{LS}}^{(k)}$ is the estimated LS failure probability corresponding to the direction $\boldsymbol{\alpha}_k$, given as:

$$p_{f,\mathsf{LS}}^{(k)} = \frac{1}{N_k} \sum_{i=1}^{N_k} \Phi(-\beta_i). \tag{1.62}$$

Here, N_k is the number of lines processed using the k-th direction. The heuristic weights w_k are calculated based on the distance from the origin to the limit-state surface along each direction, $\beta_{\boldsymbol{\alpha}_k}$, they are given as:

$$w_k = \frac{N_k \Phi(-\beta_{\boldsymbol{\alpha}_k})}{\sum_{k=1}^{K} N_k \Phi(-\beta_{\boldsymbol{\alpha}_k})}. \tag{1.63}$$

A similar approach is used for estimating the variance of the failure probability:

$$\mathrm{Var}\left[P_{f,\mathsf{LS}}\right] = \sum_{k=1}^{K} w_k^2 \cdot \mathrm{Var}\left[p_{f,\mathsf{LS}}^{(k)}\right]. \tag{1.64}$$

The weighting of the estimators is used in the LS implementation in UQLAB when the adaptive updating of the direction is enabled.

1.5.4.3 Line Sampling Algorithm

The LS algorithm in UQLAB follows these steps:

1. Identify the initial important direction $\boldsymbol{\alpha}$, using either the gradient at the origin in standard normal space or FORM.
2. Generate samples in the standard normal space and project them onto the hyperplane orthogonal to $\boldsymbol{\alpha}$.
3. Order the line processing based on the proximity of hyperplane samples (*i.e.*, line feet) to the origin.
4. Conduct root searches for each line to determine $\beta^{(i)}$, using polynomial or spline interpolation, or Newton's method.
5. If adopting the adaptive scheme and a more probable point is found, compute a new important direction, re-project remaining lines onto the new hyperplane, and re-order their processing.
6. Estimate the mean failure probability and its variance using the relevant equations: Eqs. (1.59) and (1.60) for the non-adaptive approach and Eqs. (1.61) and (1.64) for the adaptive approach, respectively.

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 UQLAB User Manual – Kriging (Gaussian process modelling)) predict the value of the limit-state function most accurately in the vicinity of the experimental design samples $\mathcal{X} = \left\{\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(n)}\right\}$. 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 $\mathcal{X} = \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N_0)}\right\}$ and evaluate the corresponding limit-state function responses $\mathcal{Y} = \left\{y^{(1)}, \ldots, y^{(N_0)}\right\} = \left\{g(\boldsymbol{x}^{(1)}), \ldots, g(\boldsymbol{x}^{(N_0)})\right\}$
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:

$$U(\boldsymbol{X}) = \frac{\left|\mu_{\widehat{g}}(\boldsymbol{X})\right|}{\sigma_{\widehat{g}}(\boldsymbol{X})}, \tag{1.65}$$

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 $\widehat{g}(\boldsymbol{x})$:

$$\boldsymbol{s}^* = \arg\min_{\boldsymbol{s}\in\mathcal{S}} U(\boldsymbol{s}) \equiv \arg\max_{\boldsymbol{s}\in\mathcal{S}} P_m(\boldsymbol{s}). \tag{1.66}$$

Another popular learning function is the *expected feasibility function* (EFF) (Bichon et al., 2008):

$$\begin{aligned} EFF(\boldsymbol{x}) = \mu_{\widehat{g}}(\boldsymbol{x}) &\left[2\Phi\left(\frac{-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \Phi\left(\frac{-\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \Phi\left(\frac{\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right)\right] \\ - \sigma_{\widehat{g}}(\boldsymbol{x}) &\left[2\varphi\left(\frac{-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \varphi\left(\frac{-\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \varphi\left(\frac{\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right)\right] \\ &+ \epsilon\left[\Phi\left(\frac{\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \Phi\left(\frac{-\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right)\right], \end{aligned} \tag{1.67}$$

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:

$$\boldsymbol{s}^* = \arg\max_{\boldsymbol{s}\in\mathcal{S}} EFF(\boldsymbol{s}). \tag{1.68}$$

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:

$$\frac{\widehat{P}_f^+ - \widehat{P}_f^-}{\widehat{P}_f^0} \leq \epsilon_{\widehat{P}_f}, \tag{1.69}$$

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

$$\widehat{P}_f^0 = \mathbb{P}\left(\mu_{\widehat{g}}(\boldsymbol{X}) \leq 0\right), \tag{1.70}$$

$$\widehat{P}_f^{\pm} = \mathbb{P}\left(\mu_{\widehat{g}}(\boldsymbol{X}) \mp k\sigma_{\widehat{g}}(\boldsymbol{X}) \leq 0\right), \tag{1.71}$$

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:

$$\frac{\widehat{\beta}^+ - \widehat{\beta}^-}{\widehat{\beta}^0} \leq \epsilon_{\widehat{\beta}}, \tag{1.72}$$

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

$$\widehat{\beta}^0 = -\Phi^{-1}(\widehat{P}_f^0), \tag{1.73}$$

$$\widehat{\beta}^{\pm} = -\Phi^{-1}(\widehat{P}_f^{\mp}). \tag{1.74}$$

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 UQLAB 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 UQLAB as an advanced option of AK-MCS (see Section 2.3.6.2).

1.6.2 Stochastic spectral embedding-based reliability

Stochastic spectral embedding-based reliability (SSER, see Wagner et al. (2021)) is an active-learning reliability method that uses the stochastic spectral embedding (SSE, see Marelli et al. (2021) and UQLAB User Manual – Stochastic spectral embedding) metamodelling technique to approximate the limit-state function with

$$g(\boldsymbol{X}) \approx g_{\text{SSE}}(\boldsymbol{X}) = \sum_{k \in \mathcal{K}\setminus\mathcal{T}} \mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^k}(\boldsymbol{X})\, \widehat{\mathcal{R}}_S^k(\boldsymbol{X}) + \sum_{k \in \mathcal{T}} \mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^k}(\boldsymbol{X})\, \widehat{\mathcal{R}}_S^k(\boldsymbol{X}), \tag{1.75}$$

where $\mathcal{T}$ is the set of indices denoting the *terminal domains*, *i.e.*the unsplit domains among all the the $\text{card}(\mathcal{K})$ domains used in the SSE representation (see Figure **??**).

Because the set of terminal domains $\mathcal{T}$ constitutes a complete partition of the input domain, *i.e.*$\mathcal{D}_{\boldsymbol{X}} = \bigcup_{k\in\mathcal{T}} \mathcal{D}_{\boldsymbol{X}}^k$, the failure probability from Eq. (1.2) can be rewritten as a weighted

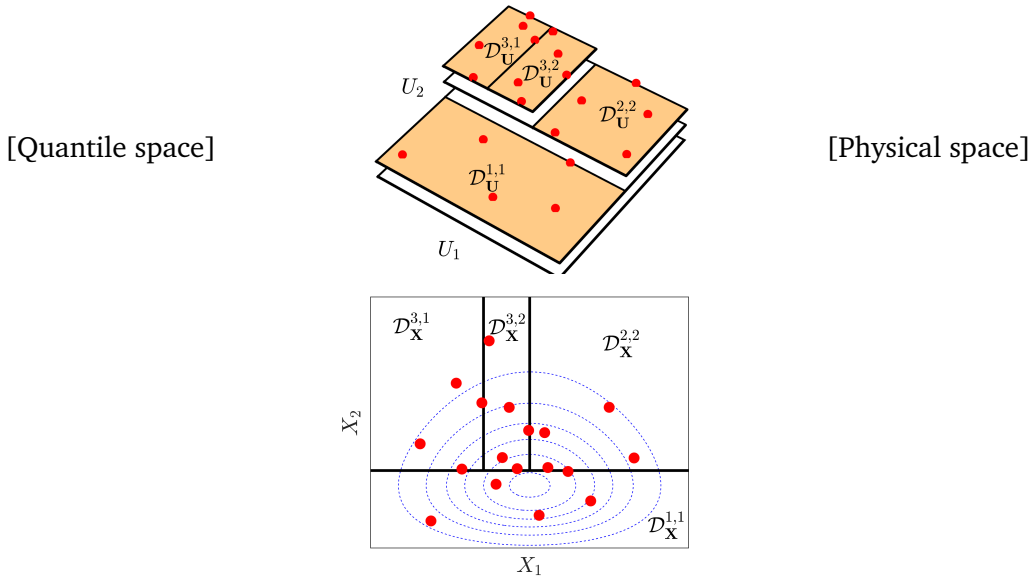

Figure 6: *Stochastic spectral embedding*: Illustration of SSE representation in Eq. (1.75) in the quantile and physical space. Red points denote the experimental design and orange areas denote terminal or unsplit domains, that partition the input space into a set of *disjoint* subdomains such that $\mathcal{D}_{\boldsymbol{X}} = \bigcup_{k\in\mathcal{T}} \mathcal{D}_{\boldsymbol{X}}^k$.

sum

$$P_f = \sum_{k\in\mathcal{T}} \mathcal{V}^k P_f^k, \quad \text{with} \quad \mathcal{V}^k = \int_{\mathcal{D}_{\boldsymbol{X}}^k} f_{\boldsymbol{X}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}, \tag{1.76}$$

where $\mathcal{V}^k$ is the *domain-wise probability mass*, and P_f^k is a *conditional failure probability* given by

$$P_f^k \overset{\text{def}}{=} \mathbb{P}\left[g(\boldsymbol{X}) \le 0 | \boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}}^k\right] \approx \mathbb{P}\left[g_{\text{SSE}}(\boldsymbol{X}) \le 0 | \boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}}^k\right]. \tag{1.77}$$

These conditional failure probabilities can then be estimated with suitable probabilistic simulation methods utilizing the SSE representation of the limit-state function.

SSER shares many similarities with other active learning reliability methods (see UQLAB User Manual – Active learning reliability). However, SSER does not strictly fit in the general active-learning reliability framework because its construction relies on the SSE-specific sequential partitioning algorithm.

1.6.2.1 Sequential partitioning algorithm

The algorithm to construct SSEs implemented in UQLAB is called *sequential partitioning algorithm*. It sequentially partitions selected refinement domains and constructs local expansions of the residual to ultimately produce a limit-state approximation of the form shown in Eq. (1.75). This algorithm is explained in detail in the UQLAB User Manual – Stochastic spectral embedding. In Wagner et al. (2021), modifications to this algorithm were proposed that turned the SSE construction into an efficient active-learning reliability algorithm. These modifications pertain to the introduction of *bootstrap SSE*, reliability-specific *refinement domain selection* and *sample enrichment* strategies as well as a reliability-specific *stopping criterion*,

and a way to estimate the *conditional failure probabilities*.

Bootstrap SSE To take into account the *point-wise* prediction accuracy of SSE, we use the local error estimation capabilities of bootstrap SSE (Efron, 1979; Marelli and Sudret, 2018; Wagner et al., 2021). Using B resampled experimental designs, we construct B residual expansions in every terminal domain to obtain B bootstrap predictions

$$g_{\text{SSE}}^{(b)}(\boldsymbol{X}) = \sum_{k\in\mathcal{K}\setminus\mathcal{T}} \mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^k}(\boldsymbol{X})\,\widehat{\mathcal{R}}_S^k(\boldsymbol{X}) + \sum_{k\in\mathcal{T}} \mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^k}(\boldsymbol{X})\,\widehat{\mathcal{R}}_S^{k,(b)}(\boldsymbol{X}), \quad \text{for} \quad b\in\{1,\cdots,B\}. \tag{1.78}$$

By analogy with Marelli and Sudret (2018), we can then define an estimate of the point-wise misclassification probability as:

$$\hat{P}_m(\boldsymbol{x}) \overset{\text{def}}{=} \frac{1}{B}\sum_{b=1}^{B}\left|\mathbf{1}_{\mathcal{D}_{f,\text{SSE}}}(\boldsymbol{x}) - \mathbf{1}_{\mathcal{D}_{f,\text{SSE}}}^{(b)}(\boldsymbol{x})\right|, \tag{1.79}$$

where $\mathbf{1}_{\mathcal{D}_{f,\text{SSE}}}$ and $\mathbf{1}_{\mathcal{D}_{f,\text{SSE}}}^{(b)}$ are evaluated with the mean and bootstrap predictors of SSE respectively.

The bootstrap replications can also be used to obtain statistics of the conditional failure probability estimates, such as variance or confidence bounds. As an example, the conditional failure probability variance is given by

$$\text{Var}\left[\hat{P}_f^k\right] = \text{Var}\left[\left\{\hat{P}_f^{k,(1)},\cdots,\hat{P}_f^{k,(B)}\right\}\right], \tag{1.80}$$

with the failure probability of the b-th replication given by

$$\hat{P}_f^{k,(b)} \overset{\text{def}}{=} \mathbb{P}\left[g_{\text{SSE}}^{(b)}(\boldsymbol{X}) \le 0 | \boldsymbol{X}\in\mathcal{D}_{\boldsymbol{X}}^k\right]. \tag{1.81}$$

By means of Eq. (1.76) this conditional failure probability variance can be used to write an expression for the total failure probability variance as

$$\text{Var}\left[\hat{P}_f\right] = \sum_{k\in\mathcal{T}}\left(\mathcal{V}^k\right)^2 \text{Var}\left[\hat{P}_f^k\right]. \tag{1.82}$$

Similarly, the bootstrap replications can be used to define *confidence bounds* on the total failure probability. Let $\hat{P}_f^{(b)} \overset{\text{def}}{=} \sum_{k\in\mathcal{T}}\mathcal{V}^k\hat{P}_f^{k,(b)}$ be the total failure probability estimated with the b-th replication from all terminal domains. The equal tail confidence bounds $\left\{\overline{\hat{P}_f^k}, \underline{\hat{P}_f^k}\right\}$ are then defined by

$$\begin{aligned}
&\mathbb{P}\left[\underline{\hat{P}_f^k} \le \left\{\hat{P}_f^{(1)},\cdots,\hat{P}_f^{(B)}\right\}\right] \approx \alpha,\\
&\mathbb{P}\left[\left\{\hat{P}_f^{(1)},\cdots,\hat{P}_f^{(B)}\right\} \le \overline{\hat{P}_f^k}\right] \approx 1-\alpha
\end{aligned} \tag{1.83}$$

with $\alpha\in[0,0.5]$ and $\gamma \overset{\text{def}}{=} 1-2\alpha$ is called the *symmetrical confidence level*.

Refinement domain selection At every refinement step, the sequential partitioning algorithm chooses a *refinement domain* based on a refinement score (see Section 1.3.1 in UQLAB User Manual – Stochastic spectral embedding). For SSER, this refinement score is given by:

$$\mathcal{E}^k = \begin{cases} \mathcal{V}^k, & \text{if Eq. (1.85) is met,} \\ \left(\mathcal{V}^k\right)^2 \mathrm{Var}\left[\hat{P}_f^k\right], & \text{otherwise,} \end{cases} \tag{1.84}$$

which depends on the following *intermediate re-prioritization criterion*

$$\frac{\mathrm{Var}\left[\left\{\hat{P}_f^{(i-2)}, \hat{P}_f^{(i-1)}, \hat{P}_f^{(i)}\right\}\right]}{\left(\hat{P}_f^{(i)}\right)^2} < \varepsilon_{\hat{P}_f}, \tag{1.85}$$

where $\hat{P}_f^{(i)}$ is the total failure probability estimator at the i-th iteration of the algorithm and the threshold is heuristically chosen to $\varepsilon_{\hat{P}_f} = 0.1\%$. This criterion is triggered when the failure probability estimator does not change significantly in three successive iterations.

Partitioning strategy The partitioning strategy determines how a selected refinement domain is split. SSER separates regions that predict failure/safety correctly from those that do so incorrectly (see Figure 7). A measure of prediction accuracy in this respect is given by the misclassification probability P_m (Echard et al., 2011) defined for bootstrap SSE in Eq. (1.79). To this end, we define two auxiliary conditional random vectors in the quantile space $\mathcal{D}_{\boldsymbol{U}}$ as

$$\boldsymbol{Z}^0 \overset{\text{def}}{=} \boldsymbol{U}|\hat{P}_m = 0, \quad \text{and} \quad \boldsymbol{Z}^+ \overset{\text{def}}{=} \boldsymbol{U}|\hat{P}_m > 0. \tag{1.86}$$

These vectors identify regions of zero ($\boldsymbol{Z}^0$) and non-zero ($\boldsymbol{Z}^+$) misclassification probability in a given refinement domain.

Based on these random vectors, we choose a set of *splitting locations* in each dimension $\upsilon_i \in \mathcal{D}_{U_i}, i = 1, \cdots, M$ that confine a maximum of $\boldsymbol{Z}^+$'s probability mass to one side of the split, while confining a maximum of $\boldsymbol{Z}^0$'s probability mass to the other side of the split. Denoting by Z_i^+ and Z_i^0 the marginals of the auxiliary random vectors in the i-th dimension, we pick the splitting location

$$\widehat{\upsilon}_i = \arg\max_{\upsilon_i \in \mathcal{D}_{X_i}} L_i(\upsilon_i), \quad \text{with} \quad L_i(\upsilon_i) \overset{\text{def}}{=} -1 + \max \begin{cases} \mathbb{P}\left[Z_i^+ \leq \upsilon_i\right] + \mathbb{P}\left[Z_i^0 > \upsilon_i\right], \\ \mathbb{P}\left[Z_i^+ > \upsilon_i\right] + \mathbb{P}\left[Z_i^0 \leq \upsilon_i\right]. \end{cases} \tag{1.87}$$

To ultimately choose a *splitting direction* $d \in \{1, \cdots, M\}$, we compare the values of the objective functions $L_i(\widehat{\upsilon}_i)$ and split along the dimension that achieves the best split, *i.e.*

$$d = \arg\max_{i \in \{1, \cdots, M\}} L_i(\widehat{\upsilon}_i). \tag{1.88}$$

In practice, a sufficiently large sample of the auxiliary random vectors $\boldsymbol{Z}^0$ and $\boldsymbol{Z}^+$ is used to conduct the computations of this section.

It might occur, at the first step of the algorithm or after the intermediate re-prioritization

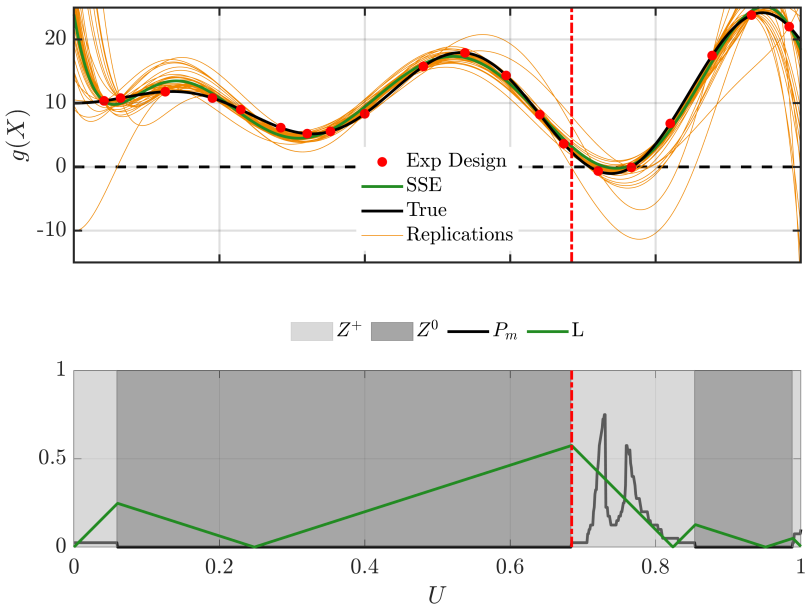

Figure 7: Partitioning strategy exemplified on a one dimensional function showing the bootstrap replications and the resulting support of Z^+ and Z^0 as well as the objective function L. The splitting location υ is highlighted as a vertical, dashed red line. The plots are shown in the quantile space $\mathcal{D}_U$.

criterion Eq. (1.85) has been triggered, that the algorithm does not detect misclassified sample points. When this happens, the auxiliary random vector $\boldsymbol{Z}^+$ is not defined. To let the algorithm proceed in this case, the definition of the auxiliary random vectors is updated in Eq. (1.86) where P_m is replaced with

$$P_t(\boldsymbol{x}) \overset{\text{def}}{=} \frac{1}{B}\sum_{b=1}^{B} \mathbf{1}_{\mathcal{D}_t}^{(b)}(\boldsymbol{x}), \tag{1.89}$$

where $\mathbf{1}_{\mathcal{D}_t}^{(b)}$ is defined with an empirical quantile t such that $\mathbb{P}\left[|g_{\text{SSE}}(\boldsymbol{X})| \leq t\right] = \varepsilon_t$ as

$$\mathbf{1}_{\mathcal{D}_t}^{(b)}(\boldsymbol{x}) \overset{\text{def}}{=} \begin{cases} 1, & \text{if} \quad |g_{\text{SSE}}^{(b)}(\boldsymbol{x})| \leq t, \\ 0, & \text{if} \quad |g_{\text{SSE}}^{(b)}(\boldsymbol{x})| > t. \end{cases} \tag{1.90}$$

The value $\varepsilon_t = 0.01$ has proven to be a good choice in practice.

Sample enrichment SSER uses a sequentially sampled experimental design (see Section 1.3.3 in UQLAB User Manual – Stochastic spectral embedding). To exploit the fact that the SSE for reliability applications needs to be more accurate near the limit-state surface, we split the sample budget and randomly place

$$N_{\text{uni}} = \max \begin{cases} \frac{N_{\text{enr}}}{2} - \frac{N_{\text{curr}}}{2} \\ 0 \end{cases} \tag{1.91}$$

points uniformly in the refinement domain quantile space, where N_{curr} is the number of existing points in the refinement domain. The remainder of the sample budget is used to randomly place sample points in the subset of the refinement domain with non-zero misclassification probability P_m. This corresponds to sampling the auxiliary random vector $\boldsymbol{Z}^+$ defined in Eq. (1.86). We do this to ensure stability of the residual expansions that are known to deteriorate in accuracy when the experimental design clusters in a small subspace.

Conditional failure probability estimation To estimate the conditional failure probabilities in Eq. (1.76), simulation methods can be used at a low computational cost. We therefore use the subset simulation method (see Section 1.5.3.3) that is known to effectively estimate even extremely small failure probabilities (Moustapha et al., 2021).
The process of estimating the conditional failure probabilities is implemented as a *post-expansion operation* (see Section 1.3.5 in UQLAB User Manual – Stochastic spectral embedding).

Stopping criterion The algorithm is terminated when the reliability index bounds, also known as *beta bounds*, do not change significantly over a set of iterations. The generalized reliability index can be computed from the failure probability as $\beta = -\Phi^{-1}(P_f)$, where Φ^{-1} is the inverse normal CDF (Ditlevsen, 1979). Using the 95% confidence intervals on $\hat{\beta}$ from the bootstrap replications as defined for the failure probability in Eq. (1.83), the stopping criterion is given by

$$\frac{\overline{\hat{\beta}} - \underline{\hat{\beta}}}{\hat{\beta}} < \varepsilon_{\hat{\beta}}, \tag{1.92}$$

where the failure threshold is heuristically set to $\varepsilon_{\hat{\beta}} = 3\%$. For stability, we abort the algorithm only if this criterion is fulfilled in three consecutive iterations.

1.6.3 Bayesian Line Sampling

Bayesian Line Sampling, originally introduced by Dang et al. (2023a,b), uses a Bayesian active learning framework in combination with line sampling. The method treats the problem of evaluating the failure probability integral as a Bayesian inference problem, allowing for the incorporation of prior knowledge and modeling of discretization error. It uses a Kriging model as the prior distribution for the distance function from the hyperplane $\beta(\boldsymbol{u}_\perp)$ and derives the posterior mean and posterior variance of the failure probability. It also introduces a learning function and a stopping criterion based on the posterior statistics of the failure probability, enabling active learning. As in adaptive line sampling, the important direction is adjusted automatically and the lines are processed in a sorted manner and iteratively. The algorithm consist of the following steps:

1. Select an initial important direction $\boldsymbol{\alpha}$ using the gradient at the origin or FORM.

2. Generate an initial observation dataset using line sampling with adaptation of the important direction. This gives the initial experimental design of the points on the hyperplane $\mathcal{U}_\perp = \left\{\boldsymbol{u}_\perp^{(1)}, \ldots, \boldsymbol{u}_\perp^{(N_0)}\right\}$ and their corresponding distances $\mathcal{B} = \left\{\beta^{(1)}, \ldots, \beta^{(N_0)}\right\}$.

3. Train a Kriging metamodel $\hat{\beta}(\boldsymbol{u}_\perp)$ based on the experimental design $\{\mathcal{U}_\perp, \mathcal{B}\}$.

4. Generate a large set of N_{MC} samples in $\boldsymbol{U}_\perp$ and evaluate the posterior mean of P_f (see Section 1.6.3.1) and variance (see Sections 1.6.3.2 and 1.6.3.3).

5. Calculate the coefficient of variation (CoV) of P_f from the mean and variance. Check whether the CoV is below a user-defined threshold. If it is, skip to Step 7, otherwise continue with Step 6.

6. Enrich the dataset by a new point $\boldsymbol{u}_\perp^*$ that maximizes the learning function defined in Eq. (1.96) or Eq. (1.98). Calculate the corresponding distance β^*

7. Return the final estimates of P_f and its CoV.

1.6.3.1 Posterior mean

The mean failure probability is estimated using the posterior Bayesian statistics of the metamodel for the distance $\hat{\beta}$

$$\hat{P}_f = \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} \Phi\left(\frac{-\hat{\beta}\left(\boldsymbol{u}_\perp^{(i)}\right)}{\sqrt{1+\sigma_{\hat{\beta}}^2\left(\boldsymbol{u}_\perp^{(i)}\right)}}\right). \tag{1.93}$$

There, $\sigma_{\hat{\beta}}^2$ denotes the variance of the Kriging model for $\hat{\beta}$. Similarly, the variance of the estimator is expressed as

$$\mathrm{Var}\left[\hat{P}_f\right] = \frac{1}{N_{MC}(N_{MC}-1)} \sum_{i=1}^{N_{MC}} \left[\Phi\left(\frac{-\hat{\beta}\left(\boldsymbol{u}_\perp^{(i)}\right)}{\sqrt{1+\sigma_{\hat{\beta}}^2\left(\boldsymbol{u}_\perp^{(i)}\right)}}\right) - \hat{P}_f\right]^2. \tag{1.94}$$

1.6.3.2 Estimation of the upper bound of the variance

In Dang et al. (2023b), the authors derive an upper bound of the standard deviation of P_f in Eq. (1.93) that serves as a measure of its epistemic uncertainty. This upper bound is given by:

$$\hat{\bar{\sigma}}_{P_f} = \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} \sqrt{\mathbb{E}_U\left[\Phi^2\left(-\left(\hat{\beta}\left(\boldsymbol{u}_\perp^{(i)}\right) + \sigma_{\hat{\beta}}\left(\boldsymbol{u}_\perp^{(i)}\right) U\right)\right)\right] - \Phi^2\left(\frac{-\hat{\beta}\left(\boldsymbol{u}_\perp^{(i)}\right)}{\sqrt{1+\sigma_{\hat{\beta}}^2\left(\boldsymbol{u}_\perp^{(i)}\right)}}\right)} \tag{1.95}$$

In this equation, U is a standard normal variable sampled to compute the expectation $\mathbb{E}_U[\cdot]$.

Based on this upper bound, a learning function is derived that aims at minimizing the epistemic uncertainty of P_f at the next iteration of the algorithm. The learning function is referred to as the *upper bound of the posterior standard deviation contribution* (UPSDC) and denoted as

$$\mathrm{UPSDC}(\boldsymbol{u}_\perp) = \sqrt{\mathbb{E}_U\left[\Phi^2\left(-\left(\hat{\beta}\left(\boldsymbol{u}_\perp\right) + \sigma_{\hat{\beta}}\left(\boldsymbol{u}_\perp\right) U\right)\right)\right] - \Phi^2\left(\frac{-\hat{\beta}\left(\boldsymbol{u}_\perp\right)}{\sqrt{1+\sigma^2_{\hat{\beta}}\left(\boldsymbol{u}_\perp\right)}}\right)} \cdot \varphi_{M-1}\left(\boldsymbol{u}_\perp\right). \tag{1.96}$$

Here, φ_{M-1} denotes the standard normal PDF in $(M-1)$ dimensions.

1.6.3.3 Estimation of the true variance

The upper bound gives a rather conservative estimate and thus might lead to a relatively large amount of samples being added in the active learning. In Dang et al. (2023a), the authors present an estimate of the actual epistemic variance

$$\hat{\sigma}^2_{P_f} = \frac{1}{N_{MC}} \sum_{j=1}^{N_{MC}} k_{\Phi(-\hat{\beta})}\left(\boldsymbol{u}_\perp^{(j)}, \boldsymbol{u}_\perp^{\prime(j)}\right) \tag{1.97}$$

where $k_{\Phi(-\hat{\beta})}$ denotes the posterior covariance function of $\Phi(-\hat{\beta})$, and $\left(\boldsymbol{u}_\perp^{(j)}, \boldsymbol{u}_\perp^{\prime(j)}\right)$ are pairs of points samples on the orthogonal hyperplane $\mathcal{H}$.
The corresponding learning function is referred to as the *posterior standard deviation contribution* (PSDC). It is obtained from the posterior variance as seen above:

$$\mathrm{PSDC}\left(\boldsymbol{u}_\perp\right) = \varphi_{M-1}\left(\boldsymbol{u}_\perp\right) \cdot \int_{\mathcal{H}} k_{\Phi(-\hat{\beta})}\left(\boldsymbol{u}_\perp, \xi\right) \varphi_{M-1}\left(\xi\right) \mathrm{d}\xi. \tag{1.98}$$

The integral is evaluated using a numerical integration scheme.

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 UQLAB.

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 UQLAB script that showcases how to deploy all of the algorithms available in the structural reliability module can be found in the example file:

`Examples/Reliability/uq_Examples_Reliability_01_RS.m`

2.2 Problem set-up

Solving a structural reliability problem in UQLAB 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 UQLAB framework is first initialized with the following command:

```
uqlab
```

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

```
MOpts.mString = 'X(:,1) - X(:,2)';                    % R-S
MOpts.isVectorized = 1;
myModel = uq_createModel(MOpts);
```

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

```
IOpts.Marginals(1).Name = 'R';                         % Resistance
IOpts.Marginals(1).Type = 'Gaussian';
IOpts.Marginals(1).Moments = [5, 0.8];
IOpts.Marginals(2).Name = 'S';                         % Stress
IOpts.Marginals(2).Type = 'Gaussian';
IOpts.Marginals(2).Moments = [2, 0.6];

myInput = uq_createInput(IOpts);
```

For more details about the configuration options available for an INPUT object, please refer to the UQLAB 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 UQLAB. 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
- SSER: Section 2.3.7
- Line Sampling: Section 2.3.8
- Bayesian Line Sampling: Section 2.3.9

2.3.1 First Order Reliability Method (FORM)

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

```
FORMOpts.Type = 'Reliability';
FORMOpts.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 8. Note that the graphical representation of the FORM iterations (right panel of Figure 8) is only produced for the 2-dimensional case.

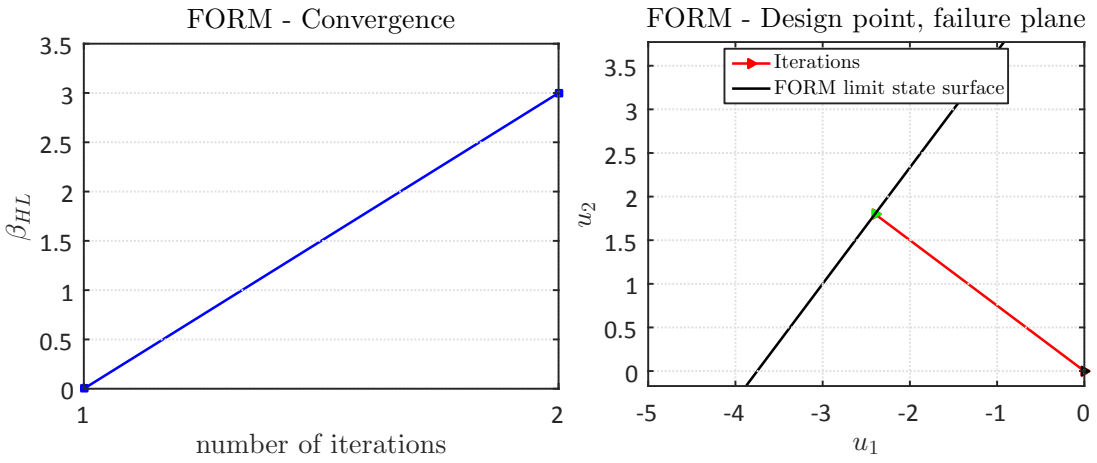

Figure 8: Graphical visualization of the results of the FORM analysis in Section 2.3.1.

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` structure:

```
FORMAnalysis.Results
ans =
               BetaHL: 3
                   Pf: 0.0013
    ModelEvaluations: 8
               Ustar: [-2.4000 1.8000]
               Xstar: [3.0800 3.0800]
          Importance: [0.6400 0.3600]
          Iterations: 2
             History: [1x1 struct]
```

In the `Results` structure, `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` field to the FORM options structure `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 $\boldsymbol{U}_0 = \mathbf{0}$. 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` structure. 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 61.

For a comprehensive list of the advanced options available to the FORM method, please see Table 6, page 61.

2.3.2 Second Order Reliability Method (SORM)

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

```
SORMOpts.Type = 'Reliability';
SORMOpts.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 8), 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` structure:

```
SORMAnalysis.Results
ans =
                Pf: 0.0013
           BetaHL: 3
  ModelEvaluations: 20
           PfFORM: 0.0013
           PfSORM: 0.0013
    PfSORMBreitung: 0.0013
            Ustar: [-2.4000 1.8000]
            Xstar: [3.0800 3.0800]
       Importance: [0.6400 0.3600]
       Iterations: 2
          History: [1x1 struct]
```

The `Results` structure contains the same fields as FORM (see Section 2.3.1) (when necessary with a `FORM` or `SORM` suffix for clarity). In addition, the two `PfSORMBreitung` and `PfSORM` fields 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';
MCOpts.Method = 'MCS';
MCOpts.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                1.3110e-03
Beta              3.0089
CoV               0.0276
ModelEvaluations  1000000
PfCI              [1.2401e-03  1.381919e-03]
BetaCI            [2.9929e+00  3.025751e+00]
-----------------------------------------
```

The results can be visualized graphically as follows:

```
uq_display(MCAnalysis)
```

which produces the convergence plots in Figure 9 and the plot of Monte Carlo sample points in Figure 10. Note that in `uq_display`, the maximum number of samples plotted is $n = 10^5$ to limit the size of the figure.

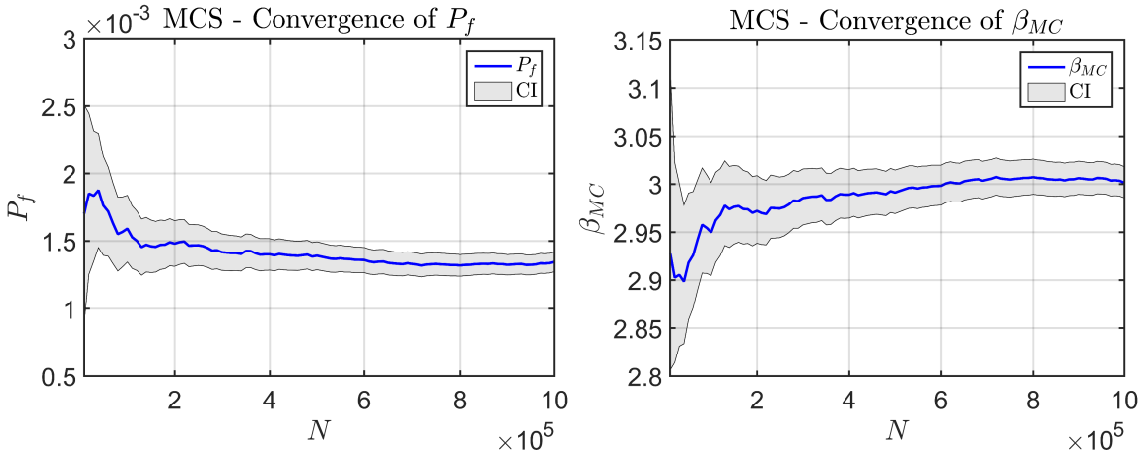

Figure 9: Graphical visualization of the convergence of the Monte Carlo simulation analysis in Section 2.3.3.

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 $g(x) \leq 0$.

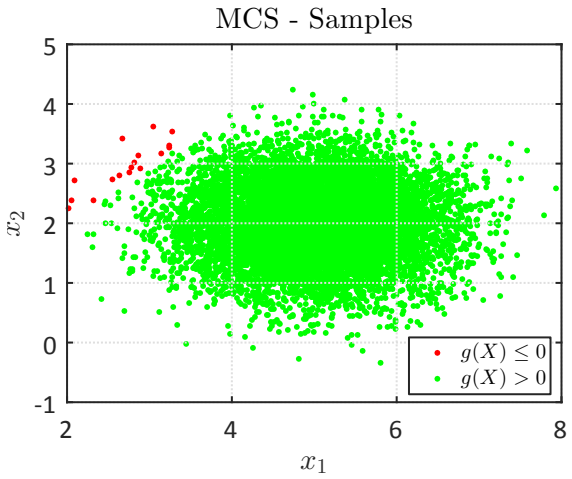

Figure 10: Graphical visualization of the samples of the Monte Carlo simulation analysis in Section 2.3.3.

2.3.3.1 Accessing the results

The analysis results can be accessed in the `MCAnalysis.Results` structure:

```
MCAnalysis.Results
ans =
                  Pf: 0.0013
                Beta: 3.0089
                 CoV: 0.0276
   ModelEvaluations: 1000000
               PfCI: [0.0012 0.0014]
             BetaCI: [2.9929 3.0258]
            History: [1x1 struct]
```

The `Results` structure contains the following fields: `Pf`, the estimated $\widehat{P}_{f,\text{MC}}$ as in Eq. (1.36); `Beta`, the corresponding generalized reliability index in Eq. (1.40); `CoV`, the coefficient of variation calculated with Eq. (1.39); `ModelEvaluations`, the total number of limit-state function evaluations; `PfCI`, the confidence intervals calculated with Eq. (1.38); `BetaCI`, the corresponding confidence intervals on β_{HL} calculated with Eq. (1.41); `History`, a structure 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` structure is used to produce the convergence plot shown in Figure 9.

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` structure. 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

`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;
MCOpts.Simulation.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` structure.

- **Specify α for the confidence intervals:** the α in Eq. (1.38) 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 60.

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';
ISOpts.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               1.3132e-03
Beta             3.0084
CoV              0.0599
ModelEvaluations 1008
PfCI             [1.1591e-03  1.467421e-03]
BetaCI           [2.9745e+00  3.046121e+00]
-----------------------------------------
```

The results can be visualized graphically as follows:

```
uq_display(ISAnalysis)
```

which produces the convergence image in Figure 11. As with FORM, the second panel in Figure 11 is produced only in the 2D case.

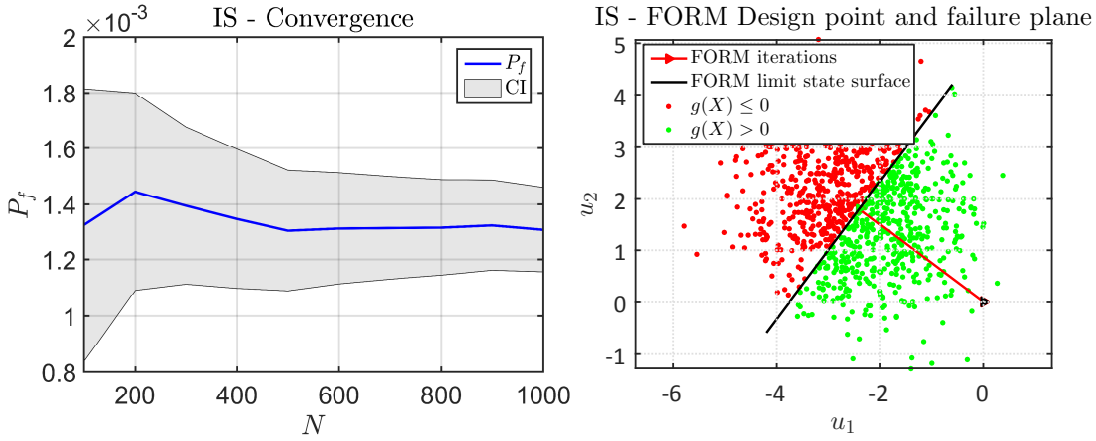

Figure 11: Graphical visualization of the convergence of the importance sampling analysis in Section 2.3.4.

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` structure:

```
ISAnalysis.Results
ans =
                Pf: 0.0013
              Beta: 3.0084
               CoV: 0.0599
  ModelEvaluations: 1008
              PfCI: [0.0012 0.0015]
            BetaCI: [2.9745 3.0461]
           History: [1x1 struct]
              FORM: [1x1 struct]
```

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

```
ISAnalysis.Results.FORM
ans =
            BetaHL: 3
```

```
                Pf: 0.0013
  ModelEvaluations: 8
             Ustar: [-2.4000 1.8000]
             Xstar: [3.0800 3.0800]
        Importance: [0.6400 0.3600]
        Iterations: 2
           History: [1x1 struct]
```

This structure 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 $P_{f,\text{IS}}$. 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` structure 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 UQLAB 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 UQLAB 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 $1 \times N_{out}$ `structure` 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';
SSOpts.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                1.0600e-03
Beta              3.0729
CoV               0.2433
ModelEvaluations  2680
PfCI              [5.5463e-04  1.565373e-03]
BetaCI            [2.9546e+00  3.261242e+00]
-----------------------------------------
```

The results can be visualized graphically as follows:

```
uq_display(SSimAnalysis)
```

which illustrates the samples of each subset in Figure 12 (applicable only for one and two-dimensional problems).

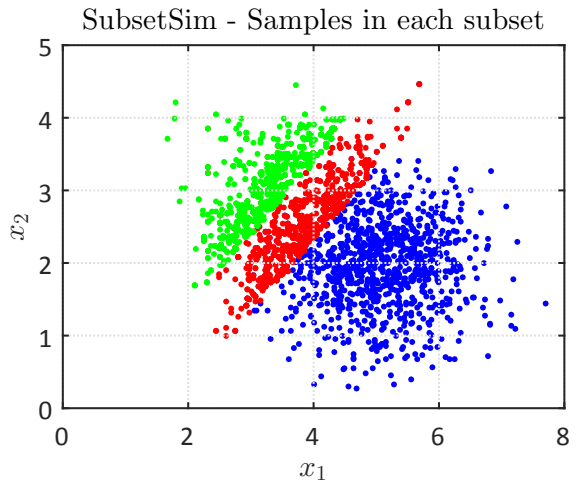

Figure 12: Graphical visualization of the convergence of the subset simulation analysis in Section 2.3.5.

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

```
SSimAnalysis.Results
ans =
                Pf: 0.0016
              Beta: 3.0729
               CoV: 0.2433
  ModelEvaluations: 2680
              PfCI: [5.5463e-04 0.0016]
            BetaCI: [2.9546 3.2612]
     NumberSubsets: 3
           History: [1x1 struct]
```

The fields in the `Results` structure have the same meaning as their counterparts in importance sampling and Monte Carlo simulation. Further, the field `NumberSubsets` denotes the number of subsets. Note that the `ModelEvaluations` field does not contain exactly the expected $N = N_S * m * (1 - P_0) = 2700$ limit-state function evaluations, but a slightly smaller $N = 2680$. 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. (1.52) 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` field. 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 62.

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 UQLAB with the following code:

```
AKOpts.Type = 'Reliability';
AKOpts.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               1.5200e-03
Beta             2.9637
CoV              0.0810
ModelEvaluations 18
PfCI             [1.2785e-03  1.761457e-03]
BetaCI           [3.0165e+00  2.917994e+00]
PfMinus/Plus     [1.5200e-03  1.520000e-03]
-----------------------------------------
```

The results can be visualized graphically as follows:

```
uq_display(AKAnalysis)
```

which produces the images in Figure 13. Note that the plot on the right of Figure 13 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.

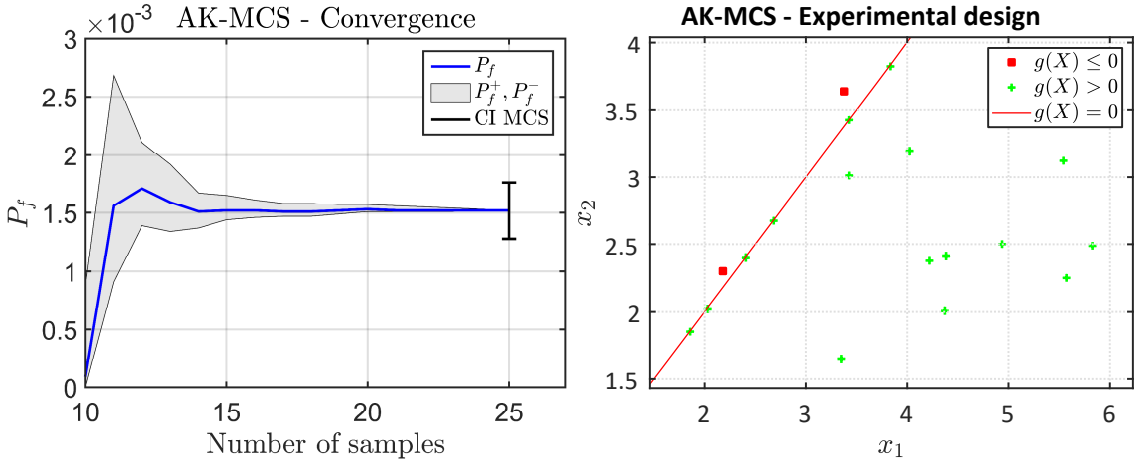

Figure 13: Graphical visualization of the convergence of the AK-MCS analysis in Section 2.3.6.

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` structure:

```
AKAnalysis.Results
ans =
                Pf: 0.0015
              Beta: 2.9637
               CoV: 0.0810
  ModelEvaluations: 18
             PfCI: [0.0013 0.0018]
           BetaCI: [3.0165 2.9180]
          Kriging: [1x1 uq_model]
          History: [1x1 struct]
```

The fields in the `Results` structure have the same meaning as their counterparts in Monte Carlo simulation. Further, the field `Kriging` contains the final Kriging metamodel used to estimate the failure probability. This metamodel can be reused within UQLAB for any other purpose, see UQLAB 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:

- **Learning function**: The learning function can be set to a custom function handle, which points to an existing MATLAB function file. For instance, for a learning function called EFF, a function called `uq_LF_EFF` must be available in the MATLAB path. It is then used in AK-MCS as:

```
AKOpts.AKMCS.LearningFunction = 'EFF';
```

- **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:

```
AKOpts.AKMCS.Convergence = 'stopPf';
```

- **Specify the Kriging metamodel**: The specifications for the Kriging metamodel (see also UQLAB User Manual – Kriging (Gaussian process modelling)) can be set in the field `.AKMCS.Kriging`, *e.g.*for an ordinary Kriging model:

```
AKOpts.AKMCS.Kriging.Trend = '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;
AKOpts.AKMCS.IExpDesign.G = G;
```

- **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 UQLAB 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 field `.AKMCS.PCK`. As an example, a Gaussian correlation function in PC-Kriging is set as:

```
AKOpts.AKMCS.PCK.Kriging.Corr.Family = 'Gaussian';
```

For an overview of the advanced options available for the AK-MCS method, refer to Table 11, page 63.

2.3.7 Stochastic spectral embedding-based reliability (SSER)

The stochastic spectral embedding-based reliability method with default values (see Table 13 and the related linked tables for details on the defaults) can be deployed in UQLAB with the following code:

```
SSEROpts.Type = 'Reliability';
SSEROpts.Method = 'SSER';
SSERAnalysis = uq_createAnalysis(SSEROpts);
```

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

```
uq_print(SSERAnalysis)
```

which produces the following:

```
--------------------------------------------------
Stochastic spectral embedding-based reliability
--------------------------------------------------
Pf                1.3686e-03
Beta              2.9958
CoV               0.0009
ModelEvaluations  80
PfCI              [1.3686e-03 1.3712e-03]
BetaCI            [2.9952e+00 2.9958e+00]
--------------------------------------------------
```

The results can be visualized graphically as follows:

```
uq_display(SSERAnalysis)
```

which produces the images in Figure 14. Note that the plot on the right of Figure 14 is only available when the input is two-dimensional.

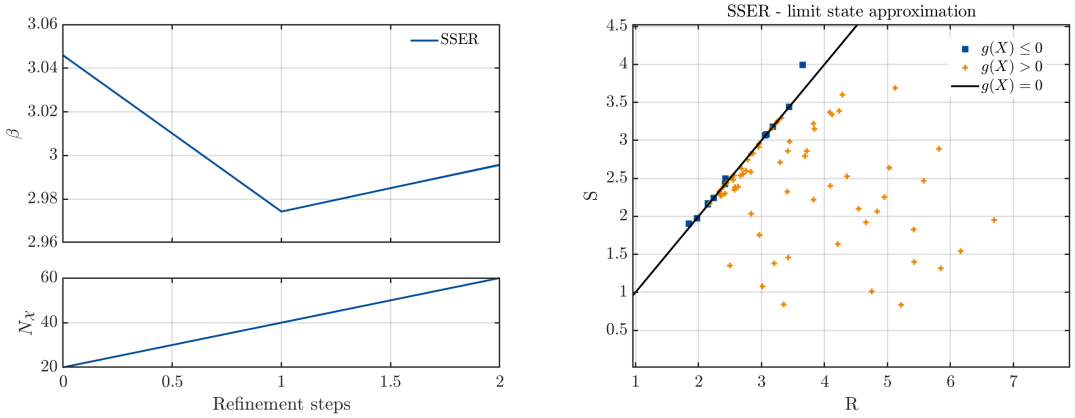

Figure 14: Graphical visualization of the convergence of the SSER analysis in Section 2.3.7.

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

- Number of enrichment samples N_{enr}: 10;
- Maximum number of evaluated samples: 10^5;
- Failure is defined for: limit-state $g(x) \leq 0$;
- Partitioning strategy: implemented in `uq_SSE_partitioning_misclass` (see Section 1.6.2.1)
- Refinement score: implemented in `uq_SSE_refineScore_Pf` (see Section 1.6.2.1)
- Post-expansion operation to estimate the conditional failure probability: implemented in `uq_SSE_postExpansion_computePf` (see Section 1.6.2.1)
- Stopping criterion: implemented in `uq_SSE_stopping_Beta` (see Section 1.6.2.1)

2.3.7.1 Accessing the results

The results from the SSER algorithm are stored in the `SSERAnalysis.Results` structure:

```
SSERAnalysis.Results

ans =

                Pf: 0.0014
              Beta: 2.9958
               CoV: 0.0009
  ModelEvaluations: 80
              PfCI: [0.0014 0.0014]
            BetaCI: [2.9952 2.9958]
              SSER: [1x1 uq_model]
           History: [1x1 struct]
```

The fields in the `Results` structure have the same meaning as their counterparts in Monte

Carlo simulation. Further, the field SSER contains the final SSE metamodel used to estimate the failure probability. This metamodel can be reused within UQLAB for any other purpose, see UQLAB User Manual – Stochastic spectral embedding for details.

For an overview of the advanced options available for the SSER method, refer to Table 13, page 65.

2.3.8 Line Sampling

Line Sampling shares some configuration options with the other methods and has several unique options. A basic LS analysis with default options is setup using the following code:

```
LSOpts.Type = 'Reliability';
LSOpts.Method = 'LS';
LSAnalysis = uq_createAnalysis(LSOpts);
```

This setup will first perform a search for the initial important direction using the gradient at the origin. Afterwards, the line sampling procedure is performed *with adaptive updating of the direction*. Once the analysis is complete, a report with the line sampling results can be printed on screen by:

```
uq_print(LSAnalysis)
```

This produces the following output in the command window:

```
  ---------------------------------------------------
  Line sampling
  ---------------------------------------------------
  Pf                1.3499e-03
  Beta              3.0000
  CoV               1.007e-16
  ModelEvaluations  2006
  PfCI              [1.3499e-03 1.3499e-03]
  BetaCI            [3.0000e+00 3.0000e+00]
  ---------------------------------------------------
```

The results can be visualized graphically as follows:

```
uq_display(LSAnalysis)
```

This command produces convergence plots for P_f and the reliability index β and in the case of a two dimensional limit-state function a figure showing the found intersection points with the limit-state surface and the samples on the hyperplane, as depicted in Figure 15. When the adaptive updating is used, the initial and final important direction are shown.

Note: In the preceding example no detailed options are provided. If not further specified, the line sampling runs with the following defaults:

- Root finder: `'Newton'`;
- Adaptive important direction: `true`;
- Initial direction: `'GradOrigin'`;
- Maximum number of lines: 1000;
- Number of lines per batch: 100;

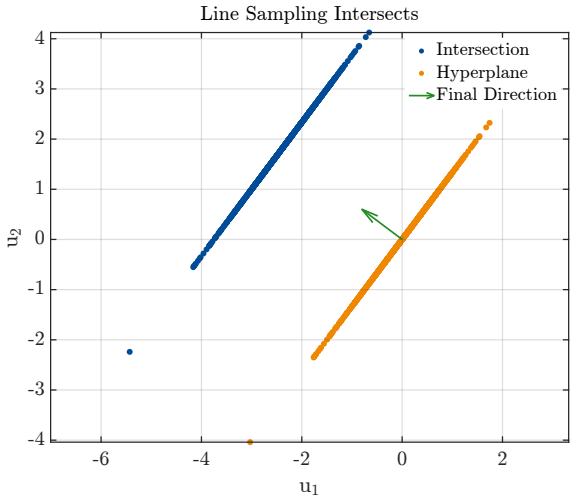

Figure 15: Graphical visualization of the lines of the line sampling analysis in Section 2.3.8.

2.3.8.1 Accessing the results

The results of line sampling are stored in the `LSAnalysis.Results` structure:

```
LSAnalysis.Results
ans =
                    Pf: 0.0013
                  Beta: 3.0000
                   CoV: 1.0067e-16
      ModelEvaluations: 2006
                 Lines: 1000
                  PfCI: [0.0013 0.0013]
                BetaCI: [3.0000 3.0000]
    ImportantDirection: [-0.8000 0.6000]
               History: [1x1 struct]
```

The fields in the `Results` structure have the same meaning as their counterparts in the other methods. `ImportantDirection` shows the final important direction that is found by the adaptive algorithm. In case of non-adaptive line sampling, this corresponds to the initial direction chosen using either the gradient of the limit-state function at the origin or FORM. The `History` struct contains more information on the updating of the direction as well as

the samples and lines processed.

2.3.8.2 Advanced options

Line sampling uses a variety of unique options as well as options for the gradient estimation and FORM, if selected. The most important ones are summarized in the following:

- **Specify the root finding algorithm:** the method used to find the intersection of the line with the limit state surface. One can set it to *e.g.* spline interpolation as follows:

```
LSOpts.LS.RootFinder.Type = 'spline';
```

- **Specify how the initial direction α is determined:** one can set it to *e.g.* FORM:

```
LSOpts.LS.Direction.Initial = 'FORM';
```

- **Specify the number of lines per batch:** one can set it to *e.g.* 50:

```
LSOpts.LS.BatchSize = 50;
```

- **Specify the maximum number of lines:** one can set it to *e.g.* 1000:

```
LSOpts.LS.MaxLines = 1000;
```

For a comprehensive overview of the available options specific to line sampling see Table 14, page 65.

2.3.9 Bayesian Line Sampling

Bayesian Line Sampling offers a metamodel-assisted line sampling analysis. It shares the options of line sampling and some unique options, most of whom are comparable to the options of AK-MCS or ALR (see UQLAB User Manual – Active learning reliability). A basic BLS analysis is performed using the following code:

```
BLSOpts.Type = 'Reliability';
BLSOpts.Method = 'BLS';
BLSAnalysis = uq_createAnalysis(BLSOpts);
```

This setup will perform a BLS analysis using the standard settings implemented in UQLAB. Once the analysis is complete, a report with the line sampling results can be printed on screen by:

```
uq_print(BLSAnalysis)
```

This produces the following output in the command window:

```
--------------------------------------------------
Bayesian line sampling
--------------------------------------------------
Pf                1.3499e-03
Beta              3.0000
```

```
CoV                2.775e-09
ModelEvaluations   28
PfCI               [1.3499e-03 1.3499e-03]
BetaCI             [3.0000e+00 3.0000e+00]
-------------------------------------------------
```

The results can be visualized graphically as follows:

```
uq_display(BLSAnalysis)
```

This produces convergence plots for P_f and the reliability index β as well as a visualization of the metamodel for the distance $\hat{\beta}$, as seen in Figure 16. Note that the latter is only available when the input is two-dimensional.

Note: In the preceding example no detailed options are provided. If not further specified, the Bayesian line sampling runs with the following defaults:

- Root finder: `'Newton'`;
- Adaptive important direction: `true`;
- Initial root: `'GradOrigin'`;
- Batch size: 10^5;
- Maximum sample size: 10^6;
- Kriging: standard options in UQLAB User Manual – Kriging (Gaussian process modelling)
- Learning Function: `'upper variance'`;
- Convergence CoV: 0.1;
- Maximum 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'`.

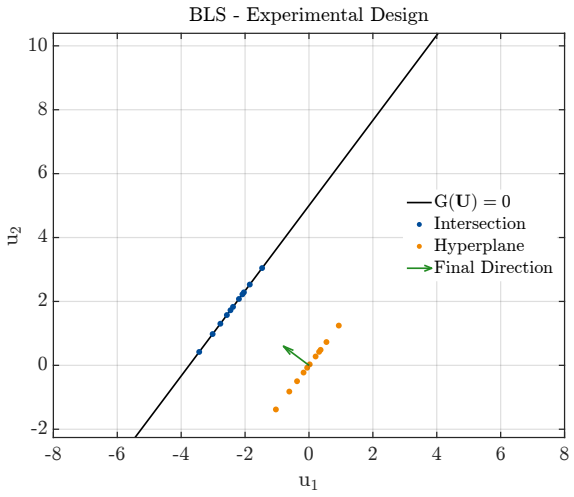

Figure 16: Graphical visualization of the Kriging model and line samples in the Bayesian line sampling analysis in Section 2.3.9.

2.3.9.1 Accessing the results

The results of line sampling are stored in the `BLSAnalysis.Results` structure:

```
BLSAnalysis.Results
ans =
                    Pf: 0.0013
                  Beta: 3.0000
                   CoV: 2.7754e-09
      ModelEvaluations: 28
                 Lines: 11
                  PfCI: [0.0013 0.0013]
                BetaCI: [3.0000 3.0000]
    ImportantDirection: [-0.8000 0.6000]
        OrthogonalBase: [0.6000 0.8000]
               History: [1x1 struct]
             Metamodel: [1x1 uq_model]
```

The fields in the `Results` structure are similar to those in line sampling. Here, the fields `OrthogonalBase` and `Metamodel` are added. The former contains the matrix needed for the transformation from the $(M-1)$-dimensional hyperplane to the M-dimensional standard normal space. This in turn is needed to visualize the Kriging MODEL for $\hat{\beta}$ stored in the field `Metamodel`.

2.3.9.2 Advanced options

Bayesian line sampling uses many of the options used in line sampling and possesses some unique options. The most important options are summarized in the following:

- **Specify the root finding algorithm:** the method used to find the intersection of the line with the limit state surface. One can set it to *e.g.* spline interpolation as follows:

```
BLSOpts.LS.RootFinder.Type = 'spline';
```

- **Specify how the initial direction α is determined:** one can set to to *e.g.* FORM:

```
BLSOpts.LS.Direction.Initial = 'FORM';
```

- **Specify the batch size:** the sample size used for the estimation of the posterior mean and variance (Eqs. (1.93) and (1.94)):

```
BLSOpts.Simulation.BatchSize = 1e4;
```

- **Specify the maximum sample size:** the maximum sample size for the estimation of the posterior mean and variance (Eqs. (1.93) and (1.94)):

```
BLSOpts.Simulation.MaxSampleSize = 1e5;
```

- **Specify the target CoV:** the target CoV for the estimation of the posterior mean and variance: the sample size is increased by batches until the CoV is smaller than the `TargetCoV` or `MaxSampleSize` has been attained.

```
BLSOpts.Simulation.TargetCoV = 0.1;
```

- **Specify the learning function:**

```
BLSOpts.BLS.LearningFunction = 'true variance';
```

- **Specify the convergence CoV:** the CoV used as the stopping criterion for the Bayesian active learning:

```
BLSOpts.BLS.ConvCoV = 0.05;
```

For a comprehensive overview of the available options specific to Bayesian line sampling see Table 17, page 67.

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 UQLAB 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;
MCOpts.LimitState.CompOp = '<=';
```

UQLAB offers several possibilities to create simple (or arbitrarily complex) objective functions from existing MODEL objects (see also UQLAB User Manual – the INPUT module).
For an overview of the advanced options for the limit-state function, refer to Table 4, page 59.

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 UQLAB *e.g.* FORM/SORM and AK-MCS, whose costs increase significantly with the number of input variables. Whenever applicable, UQLAB 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 UQLAB User Manual – the INPUT module):

```
inputOpts.Marginals.Type = 'Constant' ;
inputOpts.Marginals.Parameters = value;
```

Furthermore, when the standard deviation of a parameter equals zero, UQLAB 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

Structures 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 structures to fine-tune the inputs and outputs. Throughout this reference guide, a table-based description of the configuration structures is adopted.
The simplest case is given when a field of the structure is a simple value or array of values:

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 field. At the beginning of each row a symbol is given to inform as to whether the corresponding field 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 fields 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 one of the fields of a structure is a nested structure, a link to a table that describes the available options is provided, as in the case of the `Options` field in the following example:

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

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' ;
Input.VALUE1.Val1Opt1 = ...;
Input.VALUE1.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`	Double	Description

Table Z: `Input.VALUE2`			
☐	`.Val2Opt1`	String	Description
☐	`.Val2Opt2`	Double	Description

Note: In the sequel, `double` and `doubles` mean a real number represented in double precision and a set of such real numbers, respectively.

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 fields 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).
		`'SSER'`	Stochastic spectral embedding-based reliability (SSER).
		`'LS'`	Line sampling.
		`'BLS'`	Bayesian line sampling.
☐	`.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 field for the simulation methods. Only applies when `ROpts.Method` is `'MCS'`, `'IS'`, `'SS'`, `'AKMCS'`, or `'BLS'`.
☐	`.FORM`	See Table 6	Options field for the FORM algorithm methods. Only applies when `ROpts.Method` is `'FORM'`, `'SORM'`, `'IS'`, `'LS'`, or `'BLS'`.
☐	`.Gradient`	See Table 7	Options field for computing the gradient. It applies to the methods that use FORM, namely, when `ROpts.Method` is `'FORM'`, `'SORM'`, `'LS'`, or `'BLS'`.
☐	`.IS`	See Table 8	Options field for importance sampling. It applies only when `ROpts.Method` is `'IS'`.
☐	`.Subset`	See Table 9	Options field for subset simulation. It applies only when `ROpts.Method` is `'Subset'`.
☐	`.AKMCS`	See Table 11	Options field for the adaptive experimental design algorithm in AK-MCS. This applies when `ROpts.Method` is `'AKMCS'`.

☐	`.SSER`	See Table 13	Options field for the SSER algorithm. This applies when `ROpts.Method` is `'SSER'`.
☐	`.LS`	See Table 14	Options field for the LS algorithm. This applies when `ROpts.Method` is `'LS'` or `'BLS'`.
☐	`.BLS`	See Table 17	Options field for the BLS algorithm. This applies when `ROpts.Method` is `'BLS'`.
☐	`.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 field `ROpts.LimitState.CompOp` to `'>='`. The relevant options are summarized in Table 4:

Table 4: `ROpts.LimitState`

☐	`.Threshold`	Double 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`	Double 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`	Double	Target coefficient of variation. If present, the method will run until the estimate of the CoV (Eq. (1.39)) 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'`, `'AKMCS'`, and `'BLS'`; 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 subfield `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`	$1 \times M$ Double default: `zeros(1,M)`	Starting point for the Rackwitz-Fiessler algorithm.
☐	`.StopU`	Double 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`	Double 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`	Double 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 Struct	Instrumental distribution defined as either a structure of input marginals and copula or an INPUT object (refer to UQLAB User Manual – the INPUT module for details).
☐	`.FORM`	FORM ANALYSIS object or FORMAnalysis.Results Struct	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`	Double 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	Double 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	Structure	Kriging options when field MetaModel = 'Kriging'. If none is set, then the default Kriging options are used (refer to UQLAB 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	Structure	PC-Kriging options when field MetaModel = 'PCK'. If none is set, then the default PC-Kriging options are used (refer to UQLAB 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. (1.65)).
		'EFF'	Expected feasibility function (see Eq. (1.67)).

☐	`.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. (1.69)).
		`'stopBeta'`	Convergence criterion based on the convergence of the reliability index estimate (see Eq. (1.72)).
☐	`.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`

☐	`.Sampling`	String default: `'LHS'`	Sampling techniques of the initial experimental design. See UQLAB User Manual – the INPUT moduleFor more sampling techniques.
⊕	`.N`	Integer default: 10	Number of samples in the initial experimental design.
⊕	`.X`	$N \times M$ Double	Matrix containing the initial experimental design.
☐	`.G`	$N \times N_{out}$ Double	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 = '>='`. If `.X` is provided, but not `.G`, the limit-state responses are calculated at initialization.

SSER is an active-learning reliability method that uses a highly customized sequential partitioning algorithm to construct an SSE of the limit state function. The options for the SSE and the sequential partitioning algorithm are listed here.

Table 13: `ROpts.SSER`

☐	`.Partitioning`	Function handle default: `uq_[...]_misclass`	Partitioning strategy according to Table 8 of UQLAB User Manual – Stochastic spectral embedding.
☐	`.Refine`	Structure default: `.Score=uq_[...]_Pf`	Refinement options according to Table 11 of UQLAB User Manual – Stochastic spectral embedding.
☐	`.PostExpansion`	Function handle default: `uq_[...]_computePf`	Post-expansion operation according to Table 7 of UQLAB User Manual – Stochastic spectral embedding.

☐	`.Stopping`	Structure
		default:
		`.Criterion=...`
		`  uq_[...]_Beta`
			Stopping criterion according to Table 12 of UQLAB User Manual – Stochastic spectral embedding.

☐	`.ExpDesign`	Structure
		default:
		`.Type = 'Sequential'`
		`.Enrichment=...`
		`  uq_[...]_LSS`
		`.NEnrich=10`
		`.NSamples=2e6`
			Properties of the experimental design according to Table 18 of the UQLAB User Manual – Stochastic spectral embedding.

☐	`.ExpOptions`	Structure default: `.Boots.Repl = 200`	Properties of the residual expansions according to UQLAB User Manual – Polynomial Chaos Expansions.
☐	`.[...]`		Additional options can be selected according to Table 1 of the UQLAB User Manual – Stochastic spectral embedding.

Line sampling is a reliability method that uses lines rather than points to estimate the probability of failure. The options are listed below.

Table 14: `ROpts.LS`

☐	`.Direction`	See Table 15	Specification of the important direction α.
☐	`.RootFinder`	See Table 16	Specification of the root finder.

☐	`.MaxLines`	Integer default: 10^3	Maximum number of lines to be evaluated. If there is no target coefficient of variation (CoV), this is the total number of lines to be evaluated. If the target CoV is present, the method will run until `TargetCoV` or `MaxLines` is reached. Note that the number of maximum *samples* (and model evaluations) is dependent on the chosen root finder.
☐	`.TargetCoV`	Double default:	Target coefficient of variation. If present, the method will run until the estimate of the CoV (Eq. (1.39)) is below `TargetCoV` or until `MaxLines` 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.
☐	`.BatchSize`	Integer default: 10^2	Number of lines that will be evaluated at once.

In line sampling, the direction of the lines plays an important part in the efficiency of the algorithm. Options regarding the initial direction and updating are listed below.

Table 15: `ROpts.LS.Direction`

☐	`.Initial`	String default: `'GradOrigin'`	Method used to determine the initial important direction $\boldsymbol{\alpha}$.
		`'GradOrigin'`	Gradient at the origin in the standard normal space. For options see Table 7.
		`'FORM'`	FORM (see Section 1.4.1) with `FORM.MaxIterations` set to 2. For options see Table 6.
☐	`.Adaptive`	Logical default: true	Adaptation of the important direction once a more probable point is found.
		true	Adapt the important direction.
		false	Do not adapt the important direction.
☐	`.FORM`	FORM ANALYSIS object or `FORMAnalysis.Results` Struct	FORM results computed previously. See Section 2.3.4.2 for details.

Line sampling relies on an efficient root finder to determine the distance from the hyperplane to the limit-state surface. Options for the root finder are listed here.

Table 16: ROpts.LS.RootFinder			
☐	.Type	String default: 'Newton'	Algorithm used for the root finding.
		'Polynomial'	Perform polynomial interpolation based on the WayPoints. The degree of the polynomial is chosen as $N_{\text{WayPoints}} - 1$.
		'Spline'	Perform cubic spline interpolation based on the WayPoints.
		'Newton'	Use of Newton's method as an iterative root finder.
☐	.WayPoints	$1 \times N_{\text{WayPoints}}$ Double default: [2 5 7] for 'Polynomial'; 2:7 for 'Spline'	Way points used to perform polynomial or spline interpolation. The default polynomial interpolation is performed using 3 way points, leading to a second order polynomial fit. For spline interpolation 6 way points are chosen by default.

Bayesian line sampling offers a combination of line sampling with a Kriging metamodel for the distance and Bayesian active learning. The options of the methods are listed here.

Table 17: ROpts.BLS			
☐	.Kriging	Structure	If none is set, then the default Kriging options are used (refer to UQLAB 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.
☐	.LearningFunction	String default: 'Upper Variance'	Learning function to determine the best next sample to be added to the experimental design.
		'Upper Variance'	Use of the upper bound for variance estimation and as the subsequent learning function. See Section 1.6.3.2 for details.
		'True Variance'	Use of the true variance for estimation and as the subsequent learning function. See Section 1.6.3.3 for details.
☐	.ConvCoV	Double default: 0.1	Convergence when the calculated CoV is below the given ConvCoV.
☐	.MaxAddedED	Integer default: 1000	Maximum number of lines added to the experimental design of the Kriging metamodel.

☐	`.IExpDesign`	See Table 18	Specification of the initial experimental design of the metamodel.

The initial experimental design of BLS can be given by a number of samples and a sampling method.

Table 18: `ROpts.BLS.IExpDesign`

☐	`.N`	Integer default: $\max(10, 2 \times M)$	Number of lines in the initial experimental design.
☐	`.Sampling`	String default: `'LHS'`	Sampling techniques of the initial experimental design. See UQLAB User Manual – the INPUT module for more sampling techniques.

3.2 Accessing the results

Syntax

```
myAnalysis  = uq_createAnalysis(ROpts)
```

Output

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

- Monte Carlo - Table 19
- FORM - Table 21
- SORM - Table 23
- Importance sampling - Table 24
- Subset simulation - Table 25
- AK-MCS - Table 27
- SSER - Table 29
- Line sampling - Table 31
- Bayesian line samling - Table 34

3.2.1 Monte Carlo

The results are summarized in Table 19.

Table 19: `myAnalysis.Results`

`.Pf`	Double	Estimator of the failure probability, $P_{f,\text{MCS}}$.
`.Beta`	Double	Associated reliability index, $\beta_{\text{MCS}} = -\Phi^{-1}(P_{f,\text{MCS}})$.
`.CoV`	Double	Coefficient of variation.
`.ModelEvaluations`	Integer	Total number of model evaluations performed during the analysis.
`.PfCI`	1×2 Double	Confidence interval of the failure probability.
`.BetaCI`	1×2 Double	Confidence interval of the associated reliability index.

`.History`	See Table 20	If the simulation is carried out using batches of points, `History(i)` 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 20.

Table 20: `myAnalysis.Results.History`

`.Pf`	Double	Failure probability estimate after each batch.
`.CoV`	Double	Coefficient of variation after each batch.
`.Conf`	Double	Confidence interval after each batch.
`.X`	$N \times M$ Double	Matrix containing the input vectors in the original space evaluated by the limit-state function.
`.U`	$N \times M$ Double	Matrix containing the input vectors in the standard normal space evaluated by the limit-state function.
`.G`	$N \times N_{out}$ Double	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 structures are very similar. The results of FORM are shown in Table 21. When executing SORM, some fields will be added to the FORM `Results` structure, shown in Table 23.

Table 21: `myAnalysis.Results`

`.BetaHL`	Double	Hasofer-Lind reliability index, β_{HL}.
`.Pf`	Double	Estimator of the failure probability, $P_{f,\mathsf{FORM}} = \Phi(-\beta_{HL})$.
`.ModelEvaluations`	Integer	Total number of model evaluations performed to solve the analysis.

`.Ustar`	$1 \times M$ Double	Design point $\boldsymbol{U}^*$ in the standard normal space.
`.Xstar`	$1 \times M$ Double	Design point $\boldsymbol{X}^*$ the original space.
`.Importance`	$1 \times M$ Double	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 22	Structure with information about the algorithm steps and runtime information.

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

Table 22: `myAnalysis.Results.History`

`.ExitFlag`	String	Reason why the algorithm stopped.
`.BetaHL`	Double	Values of the reliability index along the FORM iterations.
`.OriginValue`	Double	Value of the limit-state function $G(\boldsymbol{u} = \boldsymbol{0})$.
`.G`	$N \times N_{out}$ Double	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`	Double	Values of the gradient of the limit-state function along the FORM iterations.
`.StopU`	Double	Values of the stopping criterion on the design point convergence, $\lvert\boldsymbol{U}_{k+1} - \boldsymbol{U}_k\rvert$, along the FORM iterations.
`.StopG`	Double	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`	$N \times M$ Double	Coordinates of the points $\boldsymbol{U}_k$ in the standard normal space.
`.X`	$N \times M$ Double	Coordinates of the points $\mathbf{X}_k$ in the original space.

If SORM is also performed, two fields are added to the `Results`, and two fields are added to `Results.History`, as shown in Table 23.

Table 23: `myAnalysis.Results`		
`.PfSORM`	Double	SORM estimator of the failure probability, $P_{f,\text{SORM}}$, using Hohenbichler's formula.
`.PfSORMBreitung`	Double	SORM estimator of the failure probability, $P_{f,\text{SORM}}$, using Breitung's formula.
`.BetaSORM`	Double	Associated reliability index $\beta_{\text{SORM}} = -\Phi^{-1}(P_{f,\text{SORM}})$, using Hohenbichler's formula.
`.BetaSORMBreitung`	Double	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`	$M \times M$ Double	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 structure of the results is similar to the one of Monte Carlo simulation. The results are listed in Table 24.

Table 24: `myAnalysis.Results`		
`.Pf`	Double	Estimator of the failure probability, $P_{f,\text{IS}}$.
`.Beta`	Double	Associated reliability index, $\beta_{\text{IS}} = -\Phi^{-1}(P_{f,\text{IS}})$.
`.CoV`	Double	Coefficient of variation.
`.ModelEvaluations`	Integer	Total number of model evaluations performed during the analysis.
`.PfCI`	1×2 Double	Confidence interval of the failure probability.
`.BetaCI`	1×2 Double	Confidence interval of the associated reliability index.
`.FORM`	See Table 21	Results of the FORM analysis used to find the design point.

`.History`	See Table 20	If importance sampling is carried out using batches of points, `History(i)` contains results obtained after the i-th batch.

3.2.4 Subset simulation

Since subset simulation is a simulation method, the structure of the results is similar to the one of Monte Carlo simulation. The results are listed in Table 25 and Table 26.

Table 25: `myAnalysis.Results`

`.Pf`	Double	Estimator of the failure probability, $P_{f,\mathrm{SS}}$.
`.Beta`	Double	Associated reliability index, $\beta_{\mathrm{SS}} = -\Phi^{-1}(P_{f,\mathrm{SS}})$.
`.CoV`	Double	Coefficient of variation.
`.ModelEvaluations`	Integer	Number of model evaluations during the analysis.
`.PfCI`	1×2 Double	Confidence interval of the failure probability.
`.BetaCI`	1×2 Double	Confidence interval of the associated reliability index.
`.NumberSubsets`	Integer	Number of auxiliary subsets during the analysis.
`.History`	See Table 26	Data related to each subset.

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

Table 26: `myAnalysis.Results.History`

`.delta2`	Double	δ_j^2 for estimating the coefficient of variation (see Eq. (1.54)).
`.q`	Double	Intermediate limit-state thresholds.
`.X`	Cell	Samples in the input space of each subset.
`.U`	Cell	Samples in the standard normal space of each subset.

`.G`	Cell	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`	Double	Conditional failure probability estimates $\mathbb{P}(\mathcal{D}_{i+1}\|\mathcal{D}_i)$.
`.gamma`	Double	γ_j for computing the coefficient of variation (see Eq. (1.55)).

3.2.5 AK-MCS

Since AK-MCS relies upon Monte Carlo simulation, the structure of the results is similar to the one of Monte Carlo simulation. The results are listed in Table 27 and Table 28.

Table 27: `myAnalysis.Results`

`.Pf`	Double	Estimator of the failure probability, $P_{f,\text{AK-MCS}}$.
`.Beta`	Double	Associated reliability index, $\beta_{\text{AK-MCS}} = -\Phi^{-1}(P_{f,\text{AK-MCS}})$.
`.CoV`	Double	Coefficient of variation.
`.ModelEvaluations`	Integer	Total number of model evaluations performed during the analysis.
`.PfCI`	1×2 Double	Confidence interval of the failure probability.
`.BetaCI`	1×2 Double	Confidence interval of the associated reliability index.
`.Kriging`	Struct	Final Kriging metamodel.
`.PCK`	Struct	Final PC-Kriging metamodel.
`.History`	See Table 28	Contains intermediate results along the AK-MCS iterations.

Table 28: `myAnalysis.Results.History`		
`.Pf`	Double	History of the estimate failure probability.
`.PfLower`	Double	History of the estimated lower bound of the failure probability P_f^-.
`.PfUpper`	Double	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`	Cell	Samples in the input space of each subset.
`.G`	Cell	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 = '>='`.

3.2.6 SSER

The results of an SSER are the constructed SSE and the failure probability estimate. The results are listed in Table 29 and Table 30.

Table 29: `myAnalysis.Results`		
`.Pf`	Double	Estimator of the failure probability, $P_{f,\text{SSER}}$.
`.Beta`	Double	Associated reliability index, $\beta_{\text{SSER}} = -\Phi^{-1}(P_{f,\text{SSER}})$.
`.CoV`	Double	Coefficient of variation.
`.ModelEvaluations`	Integer	Total number of model evaluations performed during the analysis.
`.PfCI`	1×2 Double	Confidence interval of the failure probability.
`.BetaCI`	1×2 Double	Confidence interval of the associated reliability index.
`.SSER`	Struct	Final SSE metamodel of the limit-state function.
`.History`	See Table 30	Contains intermediate results along the SSER iterations.

Table 30: `myAnalysis.Results.History`		
`.Pf`	Double	History of the estimate failure probability and upper and lower bounds.
`.Beta`	Double	History of the associated reliability index and upper and lower bounds.
`.CoV`	Double	History of the coefficient of variation.
`.X`	Cell	Samples in the input space that were added at a certain iteration.
`.G`	Cell	Associated values of the limit-state function.

3.2.7 Line Sampling

The structure of the results of a line sampling analysis is similar to the one of Monte Carlo simulation with some additional information regarding the lines. The results are listed in Table 31, Table 32 and Table 33.

Table 31: `myAnalysis.Results`		
`.Pf`	Double	Estimator of the failure probability, $P_{f,\mathrm{LS}}$.
`.Beta`	Double	Associated reliability index, $\beta_{\mathrm{LS}} = -\Phi^{-1}(P_{f,\mathrm{LS}})$.
`.CoV`	Double	Coefficient of variation.
`.ModelEvaluations`	Integer	Total number of model evaluations performed during the analysis.
`.Lines`	Integer	Total number of lines evaluated during the analysis.
`.PfCI`	1×2 Double	Confidence interval of the failure probability.
`.BetaCI`	1×2 Double	Confidence interval of the associated reliability index.
`.ImportantDirection`	$1 \times M$ Double	Final important direction.
`.History`	See Table 32	Contains intermediate results associated to each batch.

Table 32: `myAnalysis.Results.History`		
`.Pf`	Double	Failure probability estimate after each batch.
`.CoV`	Double	Coefficient of variation after each batch.
`.Conf`	Double	Confidence interval after each batch.

`.NLines`	Double	Number of lines in each batch.
`.NCalls`	Double	Number of model calls in each batch.
`.LineData`	See Table 33	Information on the each line.
`.X`	$N \times M$ Double	Matrix containing the input vectors in the original space evaluated by the limit-state function.
`.U`	$N \times M$ Double	Matrix containing the input vectors in the standard normal space evaluated by the limit-state function.
`.G`	$N \times N_{out}$ Double	Values of the limit-state function of each sample corrected by the threshold T (see also Table 4): *i.e.* $g(\boldsymbol{x}_k) - T$ for `Criterion = '<='`, $T - g(\boldsymbol{x}_k)$ for `Criterion = '>='`.

Table 33: `myAnalysis.Results.History.LineData`

`.ImportantDirection`	$N_{\text{Lines}} \times M$ Double	Matrix containing the important direction in the standard normal space for each line.
`.Hyperplane`	$N_{\text{Lines}} \times M$ Double	Matrix containing the coordinates in the standard normal space of the hyperplane points that initiate each line.
`.Intersects`	$N_{\text{Lines}} \times M$ Double	Matrix containing the coordinates in the standard normal space of the intersection points of each line with the limit-state surface.

3.2.8 Bayesian Line Sampling

The structure of the results of a Bayesian line sampling analysis is similar to the one of line sampling with additionally storing the Kriging metamodel. The results are listed in Table 34 and Table 35.

Table 34: `myAnalysis.Results`

`.Pf`	Double	Estimator of the failure probability, $P_{f,\text{BLS}}$.
`.Beta`	Double	Associated reliability index, $\beta_{\text{BLS}} = -\Phi^{-1}(P_{f,\text{BLS}})$.
`.CoV`	Double	Coefficient of variation.
`.ModelEvaluations`	Integer	Total number of model evaluations performed during the analysis.
`.Lines`	Integer	Total number of lines evaluated during the analysis.

.PfCI	1×2 Double	Confidence interval of the failure probability.
.BetaCI	1×2 Double	Confidence interval of the associated reliability index.
.ImportantDirection	$1 \times M$ Double	Final important direction.
.OrthogonalBase	$1 \times M$ Double	Final orthogonal base for the projection of points onto the hyperplane.
.History	See Table 35	Contains intermediate results along the Bayesian line sampling iterations.
.Metamodel	Kriging MODEL object	Contains the final Kriging model for the distance $\hat{\beta}$.

Table 35: myAnalysis.Results.History

.Pf	Double	Failure probability estimate after each iteration.
.CoV	Double	Coefficient of variation after each iteration.
.Conf	Double	Confidence interval after each iteration.
.NInit	Double	Number of lines in the initial experimental design.
.NLines	Double	Number of lines in each iteration.
.NCalls	Double	Number of model calls in each iteration.
.LineData	See Table 33	Information on the each line.
.X	$N \times M$ Double	Matrix containing the input vectors in the original space evaluated by the limit-state function.
.U	$N \times M$ Double	Matrix containing the input vectors in the standard normal space evaluated by the limit-state function.
.G	$N \times N_{out}$ Double	Values of the limit-state function of each sample corrected by the threshold T (see also Table 4): *i.e.* $g(\boldsymbol{x}_k) - T$ for Criterion = '<=', $T - g(\boldsymbol{x}_k)$ for Criterion = '>='.

3.3 Printing/Visualizing of the results

UQLAB offers two commands to conveniently print reports containing contextually relevant information for a given result object:

3.3.1 Printing the results: uq_print

Syntax

```
uq_print(myAnalysis);
uq_print(myAnalysis, outidx);
```

Description

`uq_print(myAnalysis)` prints a report on the results of the reliability analysis stored in the object `myAnalysis`. If the model has multiple outputs, only the results for the first output variable are printed.

`uq_print(myAnalysis, outidx)` prints a report on the results of the reliability analysis stored in the object `myAnalysis` for the output variables specified in the array `outidx`.

Examples:

`uq_print(myAnalysis, [1 3])` prints the reliability analysis results for output variables 1 and 3.

3.3.2 Graphically display the results: uq_display

Syntax

```
uq_display(myAnalysis);
uq_display(myAnalysis, outidx);
```

Description

`uq_display(myAnalysis)` creates a visualization of the results of the reliability analysis stored in the object `myAnalysis`, if possible. If the model has multiple outputs, only the results for the first output variable are visualized.

`uq_display(myAnalysis, outidx)` creates a visualization of the results of the reliability analysis stored in the object `myAnalysis` for the output variables specified in the array `outidx`.

Examples:

`uq_display(myAnalysis, [1 3])` will display the reliability analysis results for output variables 1 and 3.

References

Au, S. K. and Beck, J. L. (2001). Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic Engineering Mechanics*, 16(4):263–277. 13, 14, 41, 62

Bichon, B. J., Eldred, M. S., Swiler, L., Mahadevan, S., and McFarland, J. (2008). Efficient global reliability analysis for nonlinear implicit performance functions. *AIAA Journal*, 46(10):2459–2468. 18, 19

Breitung, K. (1984). Asymptotic approximations for multinormal integrals. *Journal of Engineering Mechanics*, 110(3):357–366. 10

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

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

Dang, C., Valdebenito, M. A., Faes, M. G., Song, J., Wei, P., and Beer, M. (2023a). Structural reliability analysis by line sampling: A bayesian active learning treatment. *Structural Safety*, 104:102351. 25, 27

Dang, C., Valdebenito, M. A., Song, J., Wei, P., and Beer, M. (2023b). Estimation of small failure probabilities by partially bayesian active learning line sampling: Theory and algorithm. *Computer Methods in Applied Mechanics and Engineering*, 412:116068. 25, 26

Dani, V., Hayes, T. P., and Kakade, S. M. (2008). Stochastic linear optimization under bandit feedback. In *The 21st Annual Conference on Learning Theory (COLT 2008)*. 18

de Angelis, M., Patelli, E., and Beer, M. (2015). Advanced line sampling for efficient robust reliability analysis. *Structural Safety*, 52:170–182. 16

Ditlevsen, O. (1979). Generalized second moment reliability index. *Journal of Structural Mechanics*, 7(4):435–451. 25

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

Dubourg, V. (2011). *Adaptive surrogate models for reliability analysis and reliability-based design optimization*. PhD thesis, Université Blaise Pascal, Clermont-Ferrand, France. 18

Echard, B., Gayton, N., and Lemaire, M. (2011). AK-MCS: an active learning reliability method combining Kriging and Monte Carlo simulation. *Struct. Saf.*, 33(2):145–154. 18, 19, 20, 23, 64

Efron, B. (1979). Bootstrap methods: another look at the Jackknife. *Annals of Statistics*, 7(1):1–26. 22

Ginsbourger, D., Rosspopoff, B., Pirot, G., Durrande, N., and Renard, R. (2013). Distance-based Kriging relying on proxy simulation for inverse conditioning. *Advances in Water Resources*, 52:275–291. 18

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

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

Koutsourelakis, P., Pradlwarter, H., and Schuëller, G. (2004). Reliability of structures in high dimensions, part i: algorithms and applications. *Probabilistic Engineering Mechanics*, 19(4):409–417. 15

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

Marelli, S. and Sudret, B. (2018). An active-learning algorithm that combines sparse polynomial chaos expansions and bootstrap for structural reliability analysis. *Struct. Saf.*, 75:67–74. 22

Marelli, S., Wagner, P.-R., Lataniotis, C., and Sudret, B. (2021). Stochastic spectral embedding. 11(2):25–47. 20

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

Moustapha, M., Marelli, S., and Sudret, B. (2021). A generalized framework for active learning reliability: survey and benchmark. *Reliability Engineering & System Safety*. (submitted). 25

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

Papaioannou, I. and Straub, D. (2021). Combination line sampling for structural reliability analysis. *Structural Safety*, 88:102025. 16

Rackwitz, R. and Fiessler, B. (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., Sudret, B., and Marelli, S. (2016). Rare event estimation using Polynomial-Chaos-Kriging. *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part A: Civil Engineering*, page D4016002. 18, 19, 20

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2012). Information-theoretic regret bounds for Gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265. 18

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. 11

Wagner, P.-R., Marelli, S., Papaioannou, I., Straub, D., and Sudret, B. (2021). Rare event estimation using stochastic spectral embedding. 20, 21, 22

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