UQLab User Manual Active Learning Reliability

M. Moustapha, S. Marelli, B. Sudret

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

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

M. Moustapha, S. Marelli, B. Sudret, UQLab user manual – Active learning reliability, Report UQLab-V2.1-117, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland, 2024

BibTeX entry

```
@TechReport{UQdoc_21_117,
author = {Moustapha, M. and Marelli, S. and Sudret, B.},
title = {{UQLab user manual -- Active learning reliability}},
institution = {Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich,
Switzerland},
year = {2024},
note = {Report UQLab-V2.1-117}
}
```

Document Data Sheet

Document Ref.	UQLab-V2.1-117
Title:	UQLab user manual – Active learning reliability
Authors:	M. Moustapha, S. Marelli, 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
V1.4	01/02/2021	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. A variety of methods have been developed to estimate the reliability of structures. However they are often time-consuming as they require repeated evaluations of the computational model describing the behavior of the system. Alternative and more affordable methods have been developed thanks to the introduction of surrogate models as cheap proxies of the computational model. This manual presents the UQLAB framework for active learning, a process by which the surrogate models are sequentially enriched so as to accurately estimate the failure probability with the smallest possible computational cost. The framework is made of four different components (namely surrogate model, estimation method, learning function and stopping criterion) which can be combined to build custom active learning schemes.
The active learning reliability user manual is divided in three parts:

- A short introduction to the main concepts and techniques used to solve structural reliability problems using active learning, 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 active learning reliability module.

Keywords: Active Learning, Structural Reliability, Surrogate models, Importance Sampling, Monte Carlo Simulation, Subset Simulation, AK-MCS, UQLAB, rare event estimation.

Contents

Chapter 1

Theory

1.1 Introduction

An engineering system is defined as a system required to provide specific functionality under well-defined safety constraints. Such constraints need to be taken into account during its 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 weather conditions), or in the operating conditions (*e.g.* traffic), it can occur that the system operates outside of its nominal range. In such cases, it can encounter *failure*.

Reliability analysis deals with the quantitative assessment of the probability of occurrence of such failures (a.k.a. probability of failure), given a probabilistic model of the uncertainty in the structural, environmental and load parameters.

The UQLab reliability module offers a wide range of state-of-the-art techniques for the solution of structural reliability problems, summarized in detail in the UQLab User Manual – Structural Reliability. Consistently with the overall design philosophy of UQLab, all these algorithms 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. In some cases, *e.g.* for Monte-Carlo based methods, the associated computational cost can be extremely high when the limit-state function is expensive-to-evaluate (see Section 1.2.1 for proper definitions). This document focuses on a specific class of methods, namely *active learning-based reliability methods*, whose aim is to reduce the overall computational costs of the analysis through the use of surrogate models. This is achieved by iteratively building locally accurate surrogates of the limit state function, which will then serve as a tool to explore the random variable space.

In this chapter, we first briefly recall the formalism introduced in UQLab User Manual – Structural Reliability, based on Sudret (2007). We then introduce a modular framework for active learning reliability, that allows one to easily reproduce a wide array of methods that have recently been proposed in the reliability analysis literature. This framework is made of independent blocks, which are themselves based on existing UQLab modules. The latter are interconnected non-intrusively to build custom solution strategies.

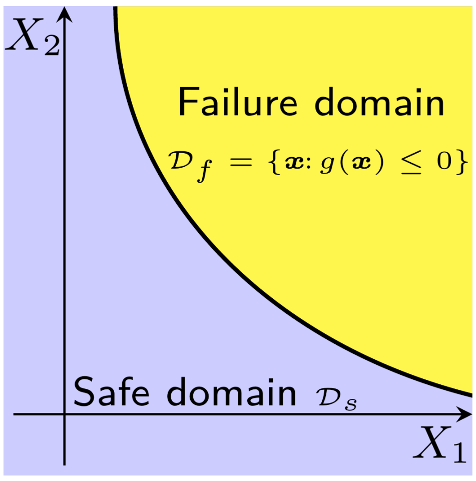
X_2
Failure domain
$\mathcal{D}_f = \{\boldsymbol{x}: g(\boldsymbol{x}) \leq 0\}$
Safe domain $\mathcal{D}_s$
X_1

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

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 of 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 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:

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

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 in $\boldsymbol{X}$ 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.

Surrogate model-based adaptive methods

Surrogate model-based adaptive methods rely on iteratively building models that approximate the limit-state function in the direct vicinity of the limit-state surface. These surrogates 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. This manual specializes in such approaches. They are thus described in details in the remainder of this chapter.

Note: This user manual focuses only on surrogate model-based adaptive methods. For more details on the other two classes of methods, please refer to the UQLAB User Manual – Structural Reliability

1.4 Surrogate-model-based strategies

1.4.1 Global framework

This module is based on the global framework for active learning reliability presented in Moustapha et al. (2021). The framework aims at building a variety of active learning schemes by non-intrusively combining methods from four different core components. These components, illustrated by a few examples, are presented in Figure 2. The first two modules, *i.e.*

Surrogate model	Reliability method	Learning function	Stopping criterion
Kriging	Monte Carlo	U	LF-based
PC-Kriging	Subset simulation	EFF	Stability of β
PCE	Importance sampling	FBR	Stability of P_f
SVR	…	CMM	Bounds on β
LRA		…	Bounds on P_f
…			…

Figure 2: Active learning reliability framework with methods available in UQLAB.

surrogate modelling and *reliability method* capitalize on readily available tools within UQLAB. They will be briefly described in the next section and the user is referred to the corresponding manual for a more detailed description. The other two, *i.e.* learning function and convergence criteria will be detailed in this manual instead.
Using these four components, an algorithm for active learning reliability can be devised and summarized as follows:

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

2. SURROGATE MODEL: Train a **surrogate model** $\widehat{g}$ (column 1 of Figure 2) on the current experimental design $\{\mathcal{X}, \mathcal{Y}\}$;

3. RELIABILITY METHOD: Using the chosen **reliability method** (column 2 of Figure 2), estimate a failure probability $\widehat{P}_f$ with the current surrogate model;

4. LEARNING FUNCTION: based on an appropriate **learning function** (column 3 of Figure 2), choose the best next sample(s) $\boldsymbol{x}^{\text{next}}$ to be added to the current experimental design $\mathcal{X}$;

5. STOPPING CRITERION: check whether a chosen **stopping criterion** (column 4 of Figure 2) is met. If it is, skip to Step 7, otherwise continue with Step 6;

6. ENRICHMENT: add $\boldsymbol{x}^{\text{next}}$ and the corresponding limit-state function response(s) $y^{\text{next}} = g(\boldsymbol{x}^{\text{next}})$ to the experimental design of the surrogate model. Return to Step 2;

7. CONCLUSION: Return the estimated failure probability $\widehat{P}_f$.

1.4.2 Surrogate modelling

Surrogate models are used in active learning reliability as a tool to explore the random space in a much affordable cost. They are built adaptively in an attempt to finely approximate the limit-state surface in areas of high probability mass.

There are various surrogate modelling approaches currently offered by UQLab. The following will be used within the active learning reliability framework:

- Gaussian process a.k.a. Kriging (UQLab User Manual – Kriging (Gaussian process modelling));
- Polynomial chaos expansions (UQLab User Manual – Polynomial Chaos Expansions);
- Polynomial chaos-Kriging (UQLab User Manual – PC-Kriging);
- Canonical low-rank tensor approximations (UQLab User Manual – Canonical low-rank approximations);
- Support vector machines for regression (UQLab User Manual – Support vector machines regression).

The user is referred to the respective manuals for more details about each of the surrogate models. Throughout this document, they will be treated as black-boxes, *i.e.* we will only be interested in their input-output structures. More specifically, for a given random variable $\boldsymbol{x}$, we denote by $\widehat{g}\left(\boldsymbol{x}\right)$ the corresponding surrogate model prediction. Note that in some cases, such as Kriging and polynomial chaos-Kriging, a second output, namely the prediction variance, is also available. This can be quite useful in active learning and will be denoted by $\widehat{s}^2\left(\boldsymbol{x}\right)$.

1.4.3 Reliability analysis algorithm

Reliability algorithms lie at the core of active learning methods, as they are primarily used to estimate the failure probability. As explained in previous sections, simulation methods are preferred to approximation ones as they often lead to more accurate results. This comes however at the expense of generally increased computational costs. When used in conjunction with surrogate-based methods, however, they can be effectively exploited, due to the extreme efficiency of surrogate model predictors, which are typically inexpensive to evaluate (often $\mathcal{O}(10^{5-6}s^{-1})$).
This module is also considered black-box and directly calls the following methods from the UQLab reliability module:

- Monte Carlo simulation, UQLab User Manual – Structural Reliability, Section 1.5.1
- Importance sampling, UQLab User Manual – Structural Reliability, Section 1.5.2
- Subset simulation, UQLab User Manual – Structural Reliability, Section 1.5.3

1.4.4 Learning function

A learning function is used to identify the points that, once added to the experimental design, would best reduce the surrogate-induced error in the estimated failure probability. The candidate point is normally chosen from a suitably defined candidate set $\mathcal{S}$, often a large

Monte-Carlo sampling of the input space, or, in this framework, the full set of limit state evaluations performed by the *reliability algorithm* of choice at the current algorithm iteration. A wide variety of learning functions have been proposed in the recent literature.
In the current version, UQLAB offers the following four:

Deviation number

Also known as U *learning function*, the *deviation number* was developed by Echard et al. (2011) based on the concept of misclassification probability, and reads:

$$U(\boldsymbol{x}) = \frac{|\widehat{g}(\boldsymbol{x})|}{\widehat{s}(\boldsymbol{x})}. \tag{1.5}$$

This quantity is directly related to the probability of misclassification in case of Gaussian distributed variables (as is the case in Kriging-based predictors) as:

$$P_m\left(\boldsymbol{x}\right) = \Phi\left(-U\left(\boldsymbol{x}\right)\right), \tag{1.6}$$

where Φ is the standard Gaussian CDF.
The next sample candidate is chosen as the one from the candidate pool $\mathcal{S}$ that maximizes the probability of misclassficiation or equivalently minimizes U:

$$\boldsymbol{x}^{\text{next}} = \arg\min_{\boldsymbol{s}\in\mathcal{S}} U\left(\boldsymbol{s}\right) = \arg\max_{\boldsymbol{s}\in\mathcal{S}} P_m\left(\boldsymbol{s}\right). \tag{1.7}$$

Efficient feasibility function

Another popular learning function was introduced by Bichon et al. (2008) and reads:

$$\begin{aligned} EFF(\boldsymbol{x}) = \widehat{g}(\boldsymbol{x}) &\left[2\Phi\left(\frac{-\widehat{g}(\boldsymbol{x})}{\widehat{s}(\boldsymbol{x})}\right) - \Phi\left(\frac{-\epsilon-\widehat{g}(\boldsymbol{x})}{\widehat{s}(\boldsymbol{x})}\right) - \Phi\left(\frac{\epsilon-\widehat{g}(\boldsymbol{x})}{\widehat{s}(\boldsymbol{x})}\right)\right] \\ &- \widehat{s}(\boldsymbol{x})\left[2\varphi\left(\frac{-\widehat{g}(\boldsymbol{x})}{\widehat{s}(\boldsymbol{x})}\right) - \varphi\left(\frac{-\epsilon-\widehat{g}(\boldsymbol{x})}{\widehat{s}(\boldsymbol{x})}\right) - \varphi\left(\frac{\epsilon-\widehat{g}(\boldsymbol{x})}{\widehat{s}(\boldsymbol{x})}\right)\right] \\ &+ \epsilon\left[\Phi\left(\frac{\epsilon-\widehat{g}(\boldsymbol{x})}{\widehat{s}(\boldsymbol{x})}\right) - \Phi\left(\frac{-\epsilon-\widehat{g}(\boldsymbol{x})}{\widehat{s}(\boldsymbol{x})}\right)\right], \end{aligned} \tag{1.8}$$

where $\epsilon = 2\widehat{s}(\boldsymbol{x})$ and φ is the PDF value of a standard normal Gaussian variable. The EFF function provides an indication of how well the true model response is expected to satisfy $g(\boldsymbol{x}) = 0$ and provides a good exploration/exploitation balance for enrichment. It takes large values when a point is close to the limit-state threshold, *i.e.* $\widehat{g}(\boldsymbol{x}) \approx 0$ and/or when there is large uncertainty in the prediction, *i.e.* $\widehat{s}(\boldsymbol{x})$ is large. The next candidate sample is then chosen as follows:

$$\boldsymbol{x}^{\text{next}} = \arg\max_{\boldsymbol{s}\in\mathcal{S}} EFF\left(\boldsymbol{s}\right). \tag{1.9}$$

Fraction of bootstrap replicates

This learning function, introduced in Marelli and Sudret (2018), uses B bootstrap replicates of the experimental design to build multiple surrogates and then eventually assess their

consistency in predicting the failed/safe state of a candidate sample. Because it is a direct measure of the misclassification probability of the current surrogate, it is closely related to the *deviation number* introduced earlier, but it does not require the predictions to be normally distributed.
It is defined as:

$$U_{\text{FBR}}\left(\boldsymbol{x}\right) = \frac{\left|B_s\left(\boldsymbol{x}\right) - B_f\left(\boldsymbol{x}\right)\right|}{B}, \tag{1.10}$$

where $B_s\left(\boldsymbol{x}\right)$ and $B_f\left(\boldsymbol{x}\right) \in [0, \ldots, B]$ are respectively the number of safe and failed bootstrap replicate predictions at the point $\boldsymbol{x}$. Samples whose prediction is uncertain tend to have close values of B_s and B_f, hence the next sample is chosen as:

$$\boldsymbol{x}^{\text{next}} = \arg\min_{\boldsymbol{s}\in\mathcal{S}} U_{\text{FBR}}\left(\boldsymbol{s}\right). \tag{1.11}$$

Constrained min-max

This learning function was introduced in Moustapha and Sudret (2019) as an adaptation of Basudhar and Missoum (2008). It aims at finding in the vicinity of the limit-state approximation samples that are the furthest from existing training points. The next sample to add then reads:

$$\boldsymbol{x}^{\text{next}} = \max_{\boldsymbol{s}\in\mathcal{S}'}\min_{i=1,\ldots,N} \|\boldsymbol{s} - \boldsymbol{x}^{(i)}\|, \tag{1.12}$$

where $\mathcal{S}' = \{\boldsymbol{s} \in \mathcal{S} : |\widehat{g}\left(\boldsymbol{s}\right)| \leq q\}$ with q being an γ-quantile of $|\widehat{g}\left(\boldsymbol{s}\right)|$. The default value $\gamma = 0.01$ is used in UQLAB.

1.4.5 Convergence criterion

Various convergence criteria can be used to terminate the enrichment process. The stopping criteria implemented in UQLAB can be split into three categories:

Variance-based criteria

These criteria relate to the surrogate-induced uncertainty in the estimates of the failure probability or in the associated reliability index $\widehat{\beta}$. This uncertainty is estimated through the built-in error measure provided by some surrogates. As such, they apply only to Kriging and PC-Kriging.
The first criterion is based on bounds of the estimated failure probability and is computed as follows:

$$\frac{\widehat{P}_f^{+} - \widehat{P}_f^{-}}{\widehat{P}_f^{0}} \leq \epsilon_{\widehat{P}_f}^{\text{bound}}, \tag{1.13}$$

where the three failure probabilities are defined as:

$$\widehat{P}_f^{0} = \mathbb{P}\left(\widehat{g}(\boldsymbol{x}) \leq 0\right), \tag{1.14}$$

$$\widehat{P}_f^{\pm} = \mathbb{P}\left(\widehat{g}(\boldsymbol{x}) \mp k\widehat{s}(\boldsymbol{x}) \leq 0\right), \tag{1.15}$$

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

The second criterion is similarly defined, however using the reliability index instead:

$$\frac{\widehat{\beta}^{+} - \widehat{\beta}^{-}}{\widehat{\beta}^{0}} \leq \epsilon_{\widehat{\beta}}^{\text{bound}}, \tag{1.16}$$

where the three reliability indices correspond to the aforementioned failure probabilities:

$$\widehat{\beta}^{0} = -\Phi^{-1}(\widehat{P}_f^{0}), \tag{1.17}$$

$$\widehat{\beta}^{\pm} = -\Phi^{-1}(\widehat{P}_f^{\mp}). \tag{1.18}$$

Stability based criteria

The second family of convergence criteria applies to all surrogate types and relates to the stability of the failure probability or reliability index estimates. They respectively read:

$$\frac{\left|\widehat{P}_f^{(j)} - \widehat{P}_f^{(j-1)}\right|}{\widehat{P}_f^{(j)}} \leq \epsilon_{\widehat{P}_f}^{\text{stab}} \tag{1.19}$$

and

$$\frac{\left|\widehat{\beta}^{(j)} - \widehat{\beta}^{(j-1)}\right|}{\widehat{\beta}^{(j)}} \leq \epsilon_{\widehat{\beta}}^{\text{stab}} \tag{1.20}$$

where $\widehat{P}_f^{(j)}$ and $\widehat{\beta}^{(j)}$ respectively represent the estimated failure probability and reliability index at the j-th iteration.

Learning function-based criteria

The final category of convergence criteria are directly based on the learning functions. For the deviation number, the criterion reads:

$$\min_{\boldsymbol{s}\in\mathcal{S}} U\left(\boldsymbol{s}\right) > \epsilon_{\text{U}}. \tag{1.21}$$

This is the original AK-MCS stopping criterion and basically means that the probability of misclassifying any point in the candidate pool is smaller than $\Phi\left(-\epsilon_{\text{U}}\right)$.
In a similar fashion, the original EFF stopping criterion is implemented and reads:

$$\max_{\boldsymbol{s}\in\mathcal{S}} EFF\left(\boldsymbol{s}\right) < \epsilon_{\text{EFF}}. \tag{1.22}$$

For the fraction of bootstrap replicates, the stopping criterion reads:

$$\min_{\boldsymbol{s}\in\mathcal{S}} U_{\text{FBR}}\left(\boldsymbol{s}\right) < \epsilon_{\text{FBR}}. \tag{1.23}$$

Finally, for the min-max criterion, the stopping criterion refers to the minimal distance to the other experimental design points

$$\max_{\boldsymbol{s}\in\mathcal{S}'} \min_{i=1,\dots,N} \|\boldsymbol{s} - \boldsymbol{x}^{(i)}\| \leq 10^{-2}\cdot d_0 \tag{1.24}$$

where d_0 is the minimum distance between the experimental design points at the first iteration.

1.5 Multiple-point enrichment

It is possible to add $K > 1$ samples within a single enrichment iteration. This is achieved in UQLAB using weighed K-means clustering (Zaki and Meira, 2014). First, a subset of the candidate pool S is defined according to the learning function:

- For U and EFF, the subset is defined as the following margin of uncertainty:

$$\mathcal{S}'' = \{\boldsymbol{x} \in \mathcal{S} : -2\,\widehat{s}\,(x) \leq \widehat{g}\,(\boldsymbol{x}) \leq 2\,\widehat{s}\,(x)\} \tag{1.25}$$

- For the fraction of bootstrap replicates, the subset is defined as:

$$\mathcal{S}'' = \{\boldsymbol{x} \in \mathcal{S} : U_{\text{FBR}}\,(\boldsymbol{x}) < 0.5\} \tag{1.26}$$

The set $\mathcal{S}''$ is then reduced into K cluster centers obtained by weighted K-means clustering using weights defined according to the learning function. They read as follows:

- Deviation number

$$w\,(\boldsymbol{x}) = \varphi\,(-U\,(\boldsymbol{x}))\,; \tag{1.27}$$

- Expected feasibility

$$w\,(\boldsymbol{x}) = \varphi\,(EFF\,(\boldsymbol{x}))\,; \tag{1.28}$$

- Fraction of bootstrap replicates

$$w\,(\boldsymbol{x}) = \varphi\,(-U_{\text{FBR}}\,(\boldsymbol{x}))\,; \tag{1.29}$$

where φ is the standard Gaussian PDF.
For the constrained min-max criterion, the enrichment is made sequentially one sample at a time, with the min-max distance updated after the addition of each sample.

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 1.

Table 1: 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

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 1 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 Basic active learning reliability

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

```
ALROpts.Type = 'Reliability';
ALROpts.Method = 'ALR';
ALRAnalysis = uq_createAnalysis(ALROpts);
```

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

```
uq_print(ALRAnalysis)
```

which produces the following:

```
-------------------------------------------
Active learning reliability
-------------------------------------------
Pf                1.4002e-03
BetaHL            2.9888
CoV               0.0189
ModelEvaluations 11
PfCI              [1.3484e-03  1.4120e-03]
BetaCI            [3.0003e+00  2.9777e+00]
PfMinus/PfPlus    [1.4002e+03  1.4002e-03]
-------------------------------------------
```

The results can be visualized graphically as follows:

```
uq_display(ALRAnalysis)
```

which produces the images in Figure 3. Note that the graphical representation of the experimental design (right panel of Figure 3) is only produced for the 2-dimensional case.

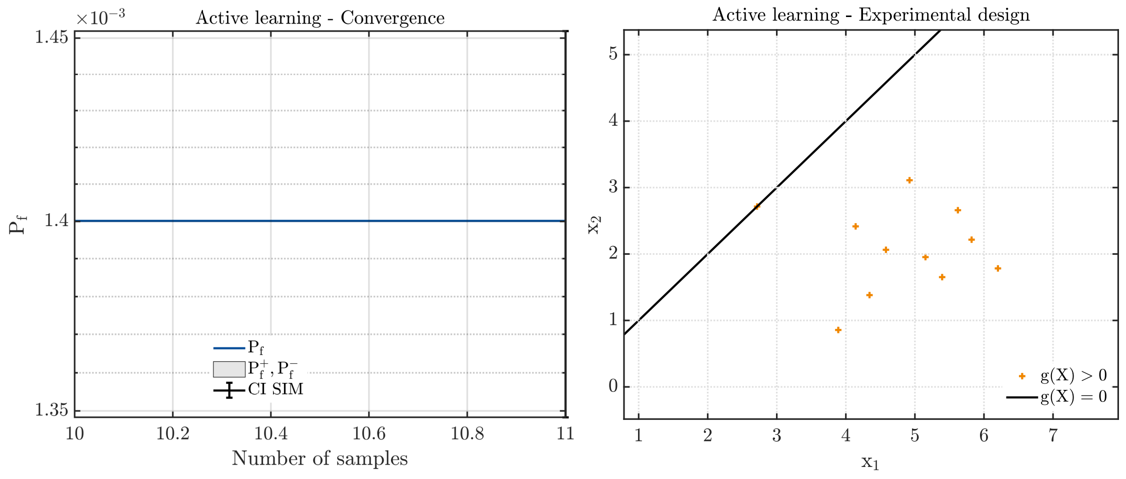

Figure 3: Graphical visualization of the results of the basic ALR analysis Section 1.4.

Note: In the preceding example, no specifications are provided. If not further specified, the ALR runs the following defaults:

- Surrogate model: PC-Kriging;
- Reliability algorithm: Subset simulation;
- Learning function : Deviation number (U-function);
- Stopping criterion: Bounds on the β estimate.

These defaults were chosen following an extensive benchmark carried out in Moustapha et al. (2021).

2.4 Accessing the results

The results from the active learning reliability analysis are stored in the `ALRAnalysis.Results` structure:

```
ALRAnalysis.Results
ans =
                  Pf: 0.0014
                 CoV: 0.0189
                Beta: 2.9888
    ModelEvaluations: 11
                PfCI: [0.0013 0.0015]
              BetaCI: [2.9777 3.0003]
           Metamodel: [1x1 uq_model]
```

```
       Reliability: [1x1 uq_analysis]
           History: [1x1 struct]
```

The `Results` structure contains the following fields: `Pf`, the estimated $\widehat{P}_f$; `Beta`, the corresponding generalized reliability index; `CoV`, the calculated coefficient of variation; `ModelEvaluations`, the total number of limit-state function evaluations; `PfCI`, the confidence intervals; `BetaCI`, the corresponding confidence intervals on β_{HL}; `History`, a structure containing the convergence of P_f, CoV, the stopping criteria values, upper and lower bounds on the failure probability (when this applies). The content of the `History` structure is used to produce the convergence plot shown in Figure 3. Note that the actual calculation of each of these results is dependent on the reliability algorithm used. The user should refer to the corresponding sections in the UQLAB User Manual – Structural Reliability for further details.

Further, the field `Metamodel` contains the final metamodel used to estimate the failure probability. This metamodel can be reused within UQLAB for any other purpose.

2.5 Advanced usage

2.5.1 Individually configure each component

In this section, we will discuss how the options for each module of the framework can be set independently.

2.5.1.1 Surrogate model

The surrogate model can be specified using the command `.ALR.Metamodel`. For instance, the user may specify the following command to use Kriging:

```
ALROpts.ALR.Metamodel = 'Kriging';
```

Furthermore, the user can specify the options of the chosen surrogate by using the command `.ALR.(MetamodelName)`, *e.g.* for Kriging:

```
KrgOpts.EstimMethod = 'ML' ;
KrgOpts.Corr.Family = 'Gaussian' ;
KrgOpts.Corr.Nugget = 1e-12;

ALROpts.ALR.Kriging = KrgOpts ;
```

Note: When no surrogate model is specified, PC-Kriging is used with the following defaults:

- PCE degree: adaptively calibrated in the range 1 to 3
- Kriging nugget: 10^{-10}
- Kriging estimation method: maximum likelihood

The remaining options are the defaults as defined in the UQLab User Manual – PC-Kriging.

All the options available for each surrogate model can be specified here without restrictions, except for the experimental design, which is handled by the ALR algorithm.

Table 2: List of supported surrogate models and user manuals

Surrogate Model	User Manual
Kriging	UQLab User Manual – Kriging (Gaussian process modelling)
PC-Kriging	UQLab User Manual – PC-Kriging
PCE	UQLab User Manual – Polynomial Chaos Expansions
LRA	UQLab User Manual – Canonical low-rank approximations
SVR	UQLab User Manual – Support vector machines regression

2.5.1.2 Reliability algorithm

To specify a reliability algorithm, the user must set the command `.ALR.Reliability`. For instance, the following code selects subset simulation:

```
ALROpts.ALR.Reliability = 'Subset' ;
```

All the options relative to the reliability algorithm can be set as defined in the UQLab User Manual – Structural Reliability. For instance, the user can specify the conditional target failure probability in subset as follows:

```
ALROpts.Subset.p0 = 0.1 ;
```

Similarly, simulation options can be defined as follows:

```
ALROpts.ALR.Simulation.BatchSize = 1e3 ;
ALROpts.ALR.Simulation.MaxSampleSize = 1e4 ;
```

Note: When no surrogate model is specified, subset simulation is used with the following options:

- Batch sample size: 10^5
- Maximum sample size: $2 \cdot 10^6$
- Conditional probability: $p_0 = 0.15$

The remaining options are the defaults as defined in the UQLAB User Manual – Structural Reliability.

2.5.1.3 Learning function

The learning function may be specified using the command `.ALR.LearningFunction`. For instance, the following line of code specifies the deviation number as learning function:

```
ALROpts.ALR.LearningFunction = 'U'
```

It is worth noting that some learning functions are available exclusively for a specific type of surrogate model. Table 3 shows all the possible combinations.

Table 3: Available combinations of surrogate models and learning function in UQLAB.

	U	EFF	FBR	CMM
Kriging	✓	✓	✗	✓
PC-Kriging	✓	✓	✗	✓
PCE	✗	✗	✓	✓
LRA	✗	✗	✗	✓
SVR	✗	✗	✗	✓

Note: When not specified by the user, the default learning function depends on the surrogate model and is chosen as follows:

- `'U'` for PC-Kriging and Kriging;
- `'FBR'` for PCE;
- `'CMM'` for SVR and LRA.

The learning function is evaluated on the candidate sample for enrichment $\mathcal{S}$, which corresponds to the samples generated during the latest run of the reliability algorithm. However, when this sample set is larger than a given value, only a subset is considered. This value is by default set to $50,000$. It can be specified using the command `LearningFunction`, e.g.,

```
   ALROpts.ALR.MaxCandidateSize = 1e5 ;
```

The subset of size `ALR.MaxCandidateSize` is obtained by random subsampling.

2.5.1.4 Stopping criterion

The stopping criterion may be specified by the user using the command `.ALR.Convergence`. The following line sets the β-bound criterion (See Eq. 1.16) as stopping criterion:

```
ALROpts.ALR.Convergence = 'StopBetaBound' ;
```

To set the corresponding threshold, say $\epsilon_{\widehat{\beta}}^{\text{bound}} = 0.05$, the following command may be used:

```
ALROpts.ALR.ConvThres = 0.05 ;
```

2.5.2 Other general options

There are several overarching options that may be needed to configure an ALR analysis. Some of the most common are summarized below:

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

  ```
  ALROpts.ALR.IExpDesign.X = X;
  ALROpts.ALR.IExpDesign.G = G;
  ```

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

  ```
  ALROpts.ALR.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.

- **Specify multiple enrichment points**: An arbitrary $K > 1$ number of points can be added simultaneously to the ED at each iteration as described in Section 1.5. For instance, $K = 3$ points can be added simultaneously using:

  ```
  ALROpts.ALR.NumOfPoints = 3;
  ```

2.5.3 Asynchronous learning

In some contexts, it is desirable to evaluate the limit state function outside the main UQLAB or even MATLAB session. We identify this process as *asynchronous learning*, because the learning algorithm and the limit state function evaluations are performed separately, possibly at different locations and times.
The ALR method supports an asynchronous learning feature, which makes it possible to perform active-learning-based reliability analysis also without defining a computational model within UQLAB. To enable this feature, the following option is needed:

```
ALROpts.Async.Enable = true ;
```

When this option is enabled, instead of evaluating the limit state function, UQLAB will interrupt the execution of the analysis and return the next sample that the user needs to evaluate to continue the algorithm. This sample is provided in the field `.Results.NextSample`. The user can then evaluate this sample outside of UQLAB or MATLAB, and provide the model evaluations `Ynext` back to UQLAB with the `uq_resumeAnalysis` command:

```
myALRAnalysis = uq_resumeAnalysis(Ynext)
```

Note: Beside the next sample that is returned, UQLAB also saves in the field `.Results.Snapshot`, a `.mat` file containing a snapshot of the analysis before it was interrupted. This can be used to restart to the analysis if the UQLAB session / MATLAB workspace was cleared or if the MATLAB session was closed. This can also be used to further investigate the current state of the analysis, *e.g.* by graphically displaying the evolution of the failure probability estimates.

2.6 Advanced limit-state function options

2.6.1 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:

```
ALROpts.LimitState.Threshold = 5;
ALROpts.LimitState.CompOp = '<=';
```

UQLAB offers several possibilities to create simple (or arbitrarily complex) objective functions from existing MODEL objects (see also UQLAB User Manual – Structural Reliability).

For an overview of the advanced options for the limit-state function, refer to Table 5, page 25.

2.6.2 Vector Outputs

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:

- at the **surrogate level**: The initial experimental design for the surrogate model of output component i consists of the final experimental design of component $i-1$.
- at the **reliability level**:
  - **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 of the limit-state.

2.7 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 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):

```
IOpts.Marginals.Type = 'Constant' ;
IOpts.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:

```
IOpts.Marginals.Type = 'Uniform' ;
IOpts.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(ALROpts)
```

Input

All the parameters required to determine the analysis are to be given as fields of the structure `ALROpts`. Each method has its own options, that will be reviewed in different tables. The options described in Table 4 are common to all methods.

Note: This manual focuses on active learning reliability methods but is generally part of the UQLAB reliability manual. All options to any reliability method can be defined in the fields of the structure `ALROpts`. However in this manual, only the options directly related to active learning will be described. For the complete set of options of the methods available in the UQLAB reliability module, the user is referred to the corresponding manual (UQLAB User Manual – Structural Reliability, Section 3).

Table 4: `ALROpts`

●	`.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:
		`'ALR'`	Active learning reliability.
		`'MCS'`	Monte Carlo simulation (without active learning).
		`'FORM'`	First order reliability method (without active learning).
		`'SORM'`	Second order reliability method (without active learning).
		`'IS'`	Importance sampling (without active learning).
		`'Subset'`	Subset simulation (without active learning).
		`'AKMCS'`	Adaptive Kriging Monte Carlo Simulation (AK-MCS).

☐	`.Name`	String	Name of the module. If not set by the user, a unique string is automatically assigned to it.
☐	`.Input`	INPUT object	INPUT object used in the analysis. If not specified, the currently selected one is used.
☐	`.Model`	MODEL object	MODEL object used in the analysis. If not specified, the currently selected one is used.
☐	`.LimitState`	See Table 5	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.
☐	`.ALR`	See Table 6	Options field for the active learning reliability scheme.
☐	`.Simulation`	See Table 7	Options field for the simulation methods used within the active learning scheme.
☐	`.IS`	See Table 8	Options field for importance sampling if used within the active learning scheme.
☐	`.Subset`	See Table 9	Options field for subset simulation if used within the active learning scheme.
☐	`.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.
☐	`.Async`	See Table 12	Options for asynchronous learning.

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 `ALROpts.LimitState.CompOp` to `'>='`. The relevant options are summarized in Table 5:

Table 5: `ALROpts.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 active learning scheme is built by choosing different methods within various modules of UQLAB, as well as some overarching options such as the initial experimental design specification. In Table 6, the available options are shown:

Table 6: `ALROpts.ALR`

☐	`.Metamodel`	String default: `'PCK'`	Surrogate model type to use in the active learning scheme (as described in Section 1.4.2).
		`'PCK'`	Polynomial Chaos - Kriging.
		`'Kriging'`	Kriging.
		`'PCE'`	Polynomial chaos expansions.
		`'SVR'`	Support vector machines for regression.
		`'LRA'`	Low rank approximation.
☐	`.Reliability`	String default: `'Subset'`	Reliability algorithm used in the active learning scheme (as described in Section 1.4.3).
		`'MCS'`	Monte Carlo simulation.
		`'Subset'`	Subset simulation.
		`'IS'`	Importance sampling.

☐	.LearningFunction	String or cell of strings	Learning function used in the active learning scheme (as described in Section 1.4.4). The default depends on the surrogate defined in .ALR.Metamodel and is set as: • 'U' when using 'PCK' or 'Kriging', • 'FBR' when using 'PCE', • 'CMM' when using 'SVR' or 'LRA'.
		'U'	Deviation number as defined in Eq. (1.5) • Only valid when .ALR.Metamodel is set to 'PCK' or 'Kriging'.
		'EFF'	Expected feasibility function as defined in Eq. (1.8). • Only valid when .ALR.Metamodel is set to 'PCK' or 'Kriging'.
		'FBR'	Fraction of bootstrap replicates as defined in Eq. (1.10). • Only valid when .ALR.Metamodel is set to 'PCE'.
		'CMM'	Constrained min-max as defined in Eq. (1.12).
☐	.Convergence	String default: 'StopBetaBound' when .ALR.Metamodel is 'Kriging' or 'PCK' and 'StopBetaStab' otherwise	Convergence criterion used in the active learning scheme (as described in Section 1.4.5).
		'StopBetaBound'	Convergence criterion based on the accuracy of the reliability index estimate w.r.t. the surrogate error (See Eq. (1.16)). • Only valid when .ALR.Metamodel is set to 'PCK' or 'Kriging'.
		'StopPfBound'	Convergence criterion based on the accuracy of the failure probability estimate w.r.t. the surrogate error (See Eq. (1.13)). • Only valid when .ALR.Metamodel is set to 'PCK' or 'Kriging'.
		'StopBetaStab'	Convergence criterion based on the stability of the reliability index estimate (See Eq. (1.20)).

		`'StopPfStab'`	Convergence criterion based on the stability of the failure probability estimate (See Eq. (1.19)).
		`'StopLF'`	Convergence criterion based on the learning function defined in `.ALR.Learning`. • For `'U'`, see Eq. (1.21). • For `'EFF'`, see Eq. (1.22). • For `'FBR'`, see Eq. (1.23). • For `'CMM'`, see Eq. (1.24).
☐	`.ConvThres`	Double	Threshold for the convergence criterion defined in `.ALR.Convergence`. Default is: • 2 for ϵ_{U} in Eq. (1.21) • 10^{-3} for ϵ_{EFF} in Eq. (1.22) • 10^{-2} for the remaining stopping criteria • When multiple convergence criteria are used, this can be either a scalar (then the same threshold is used for all criteria) or a vector of size $1 \times N_{crit}$ where N_{crit} is the number of convergence criteria defined.
☐	`.NumOfPoints`	Integer default: 1	Number of points to add within an enrichment step (See Section 1.5 for multiple point enrichment).
☐	`.ConvIter`	Integer default: 2	Number of successive iterations the convergence criteria should be satisfied for the algoritm to stop.
☐	`.MaxAddedED`	Integer default: 1000	Maximum number of samples to add to the experimental design of the metamodel.
☐	`.IExpDesign`	See Table 11	Specification of the initial experimental design of the metamodel.
☐	`.MaxCandidateSize`	Integer default: 50000	Maximum number of samples in the candidate set for enrichment. If the initial candidate size is larger than this number, random subsampling is performed to reduce its size to `.ALR.MaxCandidateSize`.

The available methods to perform structural reliability analysis within an active learning scheme are Monte Carlo simulation, importance sampling and subset simulation. They all share the simulation options which are shown in Table 7:

Table 7: `ALROpts.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: $2 \cdot 10^6$ for `'SuS'`; 10^5 for `'IS'`; 10^7 for `'MCS'`	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: 10^6 for `'MCS'`; 10^5 for `'Subset'`; 10^4 for `'IS'`	Number of samples that will be evaluated at once.

The options specifically set for the importance sampling are presented in Table 8. Note that the options of `.Simulation` and `.FORM` (not shown in this manual) are also processed in the case of importance sampling due to the nature of the MCS, FORM and IS.

Table 8: `ALROpts.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: `ALROpts.Subset`

☐	`.p0`	Double default: 0.15	Target conditional failure probability of auxiliary limit-states $(0 < \texttt{p0} \leq 0.5)$.
☐	`.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: `ALROpts.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.

The initial experimental design of surrogate model 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 11: `ALROpts.ALR.IExpDesign`

☐	`.Sampling`	String default: `'LHS'`	Sampling techniques of the initial experimental design. See UQLAB User Manual – the INPUT module for more sampling techniques.
⊕	`.N`	Integer default: $\max(10, 2 \times M)$	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 5): $(g(\boldsymbol{x}) - T)$ for `Criterion = '<='`, $(T - g(\boldsymbol{x}))$ for `Criterion = '>='`. If `.X` is provided, but not `.G`, the limit-state responses are calculated at initialization.

Asynchronous learning can be enabled when a computational model is not directly associated to the analysis. In this case, the analysis is interrupted and the next sample to add to the experimental design is returned.

Table 12: `ALROpts.Async`

☐	`.Enable`	Logical default: false	Enabling or not the asynchronous learning option.
		true	Asynchronous learning is enabled.
		false	Asynchronous learning is disabled.
☐	`.InitED`	Logical default: false	Enabling or not the asynchronous learning option for the initial experimental design.
		true	Asynchronous learning is enabled for the initial experimental design.
		false	Asynchronous learning is disabled for the initial experimental design.

3.2 Accessing the results

Syntax

```
myAnalysis  = uq_createAnalysis(ALROpts)
```

Output

The results are summarized in Table 13.

Table 13: `myAnalysis.Results`

`.Pf`	Double	Estimator of the failure probability, $\widehat{P}_f$.
`.Beta`	Double	Associated reliability index, $\widehat{\beta} = -\Phi^{-1}(\widehat{P}_f)$.
`.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.
`.Metamodel`	`uq_model`	Final surrogate model used in the analysis.
`.Reliability`	`uq_analysis`	Final reliability analysis carried out.
`.History`	See Table 14	History of the active learning reliability analysis.
`.Xnext`	$K \times M$ Double	Next sample(s) to evaluate to add to the experimental design. • Only available when `.Async.Enable` is `true`.
`.Snapshot`	String	Path to the `.mat` file containing a snapshot of the analysis. • Only available when `.Async.Enable` is `true`.

The history of the active learning process is saved in the `.History` field. Its contents is described in Table 14.

Table 14: `myAnalysis.Results.History`

`.Pf`	Double	Failure probability estimate before each enrichment.
`.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.
`.NCurrent`	Integer	History of the experimental design size.
`.NInit`	Integer	Number of samples in the initial experimental design.
`.X`	$N \times M$ double	Samples in the experimental design for the given output.
`.G`	$N \times N_{out}$ double	Values of the limit-state function of the experimental design samples corrected by the threshold T (see also Table 5): $(g(\boldsymbol{x}) - T)$ for `Criterion = '<='`, $(T - g(\boldsymbol{x}))$ for `Criterion = '>='`.
`.ReliabilitySample`	Double	Samples used in the latest reliability analysis.
`.Convergence`	Double	History of the convergence criterion values.

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. 29

Basudhar, A. and Missoum, S. (2008). An improved adaptive sampling scheme for the construction of explicit boundaries. *Struct. Multidisc. Optim.*, 42(4):517–529. 8

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. 7

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

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. 7

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. 7

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

Moustapha, M., Marelli, S., and Sudret, B. (2021). A generalized framework for active learning reliability: survey and benchmark. *Reliability Engineering & System Safety*. (submitted). 4, 13

Moustapha, M. and Sudret, B. (2019). Surrogate-assisted reliability-based design optimization: a survey and a unified modular framework. *Struct. Multidisc. Optim.*, 60:1–20. 8

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

Zaki, M. J. and Meira, W. J. (2014). *Data Mining and Analysis: fundamental Concepts and Algorithms*. Cambridge University Press. 10