ETH zürich

UQLab User Manual
Reliability-Based Design Optimization

M. Moustapha, S. Marelli, 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

M. Moustapha, S. Marelli, B. Sudret, UQLab user manual – Reliability-Based Design Optimization, Report UQLab-V2.0-115, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland, 2022

BIBTEX entry

```
@TechReport{UQdoc_20_115,
author = {Moustapha, M., Marelli, S., and Sudret, B.},
title = {{UQLab user manual -- Reliability-Based Design Optimization}},
institution = {Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich,
Switzerland},
year = {2022},
note = {Report UQLab-V2.0-115}
}
```

Document Data Sheet

Document Ref.	UQLAB-V2.0-115
Title:	UQLAB user manual – Reliability-Based Design Optimization
Authors:	M. Moustapha, S. Marelli, B. Sudret Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland
Date:	01/02/2022

Doc. Version	Date	Comments
V2.0	01/02/2022	UQLAB V2.0 release
V1.4	01/02/2021	UQLAB V1.4 release
V1.3	19/09/2019	Initial release

Abstract

Reliability-based design optimization (RBDO) is a popular approach for the optimal design of complex systems under uncertainties. The optimal design is found by trading the cost of the system with its probability of failure which accounts for uncertainties associated to its design and environmental or operating conditions. A wide variety of methods have been developed throughout the years in this active research field.
The RBDO module of UQLAB offers a wide set of techniques for the efficient solution of RBDO problems. These are mainly surrogate- and simulation-based double-loop techniques and state-of-the-art approximation methods.
The RBDO user manual is divided in three parts:

- A short introduction to the main concepts and techniques used to solve RBDO 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 RBDO module.

Keywords: Reliability-based design optimization, RBDO, structural reliability, double-loop approach, surrogate modelling, failure probability, quantile.

Contents

Chapter 1

Theory

1.1 Introduction

The optimal design of complex systems is a necessary task in a challenging industrial environment which requires finding an optimal balance between performance and cost. Structural design optimization aims at minimizing a given cost while ensuring that the structure performs under well-defined safety constraints. In the presence of uncertainties that may affect either their design parameters (*e.g.* manufacturing tolerances) or their operating conditions (*e.g.* loadings), it is possible that systems designed under deterministic considerations operate outside their nominal range. It is therefore necessary to account for uncertainties in the early design stages. Very often uncertainties are either overlooked or accounted for implicitly through semi-probabilistic design methods, *e.g.* the partial safety factor approach. For an explicit consideration of uncertainties in the design of structures, *reliability-based design optimization* (RBDO) is commonly considered.

The RBDO solution is basically achieved by jointly performing a reliability analysis and solving an optimization problem. According to the technique used for reliability analysis, RBDO schemes can be classified into approximation- and simulation-based methods (Valdebenito and Schuëller, 2010; Moustapha and Sudret, 2019). The former relies on approximation techniques such as the first-order reliability method (FORM) to solve the reliability analysis. Classical RBDO methods fall within this category. The most popular ones are implemented in UQLAB. These include the *reliability index approach* (RIA), the *performance measure approach* (PMA), the *sequential optimization and reliability assessment* (SORA) and the *single loop approach* (SLA) (Chateauneuf and Aoues, 2008). In its general setting though, UQLAB focuses on directly solving the RBDO problem rather than using approximation techniques. The incurred computational cost is leveraged by introducing *surrogate models*. The practical implementation follows the surrogate-assisted RBDO framework presented in Moustapha and Sudret (2019).

1.2 Problem statement

1.2.1 General formulation of a RBDO problem

In the set up of a reliability-based design optimization problem, one aims at reducing the cost of the system while complying with some performance constraints that account for various uncertainties. A general formulation of the problem reads (Dubourg et al., 2011):

$$\boldsymbol{d}^* = \underset{\boldsymbol{d}\in\mathbb{D}}{\arg\min}\, \mathfrak{c}\left(\boldsymbol{d}\right) \quad \text{subject to:} \quad \begin{cases} \mathfrak{f}_j\left(\boldsymbol{d}\right) \leq 0, & \left\{j = 1,\ldots,s\right\}, \\ \mathbb{P}\left(g_k\left(\boldsymbol{X}\left(\boldsymbol{d}\right),\boldsymbol{Z}\right) \leq 0\right) \leq \bar{P}_{f_k}, & \left\{k = 1,\ldots,n\right\}. \end{cases} \tag{1.1}$$

In this setting, the cost function $\mathfrak{c}$ is minimized with respect to the *design variables* $\boldsymbol{d} \in \mathbb{D} \subset \mathbb{R}^{M_d}$. The so-called *soft constraints* $\mathfrak{f}_j, j = \{1,\ldots,s\}$ are simple functions that bound the design space. In contrast, the *hard constraints* $g_k, k = \{1,\ldots,n\}$ are limit-state functions which describe the performance of the system. They are most often based on a computational model $\mathcal{M}$ that describes the system's performance. The system is considered to be in a failure state when $g_k\left(\boldsymbol{x}\left(\boldsymbol{d}\right),\boldsymbol{z}\right) \leq 0$ for any design $\boldsymbol{d}$, where $\boldsymbol{x}$ and $\boldsymbol{z}$ are realizations of the random variables $\boldsymbol{X} \in \mathbb{R}^{M_d} \sim f_{\boldsymbol{X}|\boldsymbol{d}}$ and $\boldsymbol{Z} \in \mathbb{R}^{M_z} \sim f_{\boldsymbol{Z}}$, respectively. The former is a set of random variables indexed on the design variables which may represent manufacturing tolerances while the latter are the so-called *environmental variables* which are parameters that may be random but cannot be controlled by the designer, *e.g.* the loading. This formulation is general and allows one to consider special cases when either or both the design and environmental variables are random. In RBDO, the probabilistic constraints require the failure probability $\mathbb{P}\left(g_k\left(\boldsymbol{X}\left(\boldsymbol{d}\right),\boldsymbol{Z}\right) \leq 0\right)$ to be lower than a given threshold, herein $\bar{P}_{f_k}$. For each constraint, this failure probability reads:

$$P_{f_k}\left(\boldsymbol{d}\right) = \mathbb{P}\left(g_k\left(\boldsymbol{W}\right) \leq 0\right) = \int_{g_k(\boldsymbol{w})\leq 0} f_{\boldsymbol{W}}\left(\boldsymbol{w}\right) d\boldsymbol{w}, \tag{1.2}$$

where $\boldsymbol{W} = \left\{\boldsymbol{X},\boldsymbol{Z}\right\}^T \in \mathbb{R}^M \sim f_{\boldsymbol{W}}$ is a vector gathering the design and environmental variables.

1.2.2 An alternative formulation using the reliability index

In the above formulation, constraints are expressed in terms of failure probabilities, the values of which may be extremely small and vary across several orders of magnitude. This might cause numerical issues for some optimization algorithms. An alternative expression of the constraints can be obtained by considering the *reliability index* as an equivalent expression of the failure probability (Hasofer and Lind, 1974; Marelli et al., 2021). This then leads to the following optimization problem:

$$\boldsymbol{d}^* = \underset{\boldsymbol{d}\in\mathbb{D}}{\arg\min}\, \mathfrak{c}\left(\boldsymbol{d}\right) \quad \text{subject to:} \quad \begin{cases} \mathfrak{f}_j\left(\boldsymbol{d}\right) \leq 0, & \left\{j = 1,\ldots,s\right\}, \\ \bar{\beta}_k - \beta_k\left(\boldsymbol{X}\left(\boldsymbol{d}\right),\boldsymbol{Z}\right) \leq 0, & \left\{k = 1,\ldots,n\right\}, \end{cases} \tag{1.3}$$

where $\bar{\beta}_k = \Phi^{-1}\left(1-\bar{P}_{f_k}\right)$ and $\beta_k = \Phi^{-1}\left(1-P_{f_k}\right)$ are the target and structural reliability indices of the k-th limit-state, respectively and Φ is the standard Gaussian cumulative distri-

bution function.

Note: In general, when using failure probabilities directly, UQLAB considers the probabilistic constraint in the $\log_{10}$ space. This option can be disabled manually.

1.3 Solution of a RBDO problem

There exists a wide variety of methods to solve the problem stated in Eq. (1.1). They have been classified into three groups, namely *two-level*, *mono-level* and *decoupled* approaches (Chateauneuf and Aoues, 2008). This chapter briefly reviews the corresponding methods that are implemented in UQLAB.

1.3.1 Two-level approach

The two-level approach is one of the most common techniques for solving the RBDO problem formulated in Section 1.2. It consists of nested loops where the outer loop explores the design space while the inner one computes the corresponding failure probability $P_{f_k}\left(\boldsymbol{d}^{(i)}\right)$.
This approach is implemented in UQLAB by using the reliability module in the inner loop (See UQLAB User Manual – Structural Reliability). It is referred to as the *generalized two-level* approach in this manual. Two classical approaches in the RBDO literature are also implemented, namely the *reliability index approach* and the *performance measure approach*. They can be seen as special cases of the generalized two-level approach.

1.3.1.1 Generalized two-level approach

In the generalized two-level approach, a general-purpose optimization algorithm is run for the solution of Eq. (1.1). In each iteration, the probabilistic constraint is evaluated through the estimation of the failure probability as expressed in Eq. (1.2). UQLAB offers various strategies to achieve this using either approximation or simulation techniques.

Approximation methods

In approximation methods, the limit-state function is locally approximated at a reference point (*e.g.* with a linear or quadratic Taylor expansion) thus allowing for an easier expression of the failure probability. This class of methods can be extremely efficient (in that only a relatively small number of model evaluations is needed to compute P_f) but they tend to become unreliable in the presence of complex, non-linear limit-state functions. Two approximation methods are currently available in UQLAB:

- **First-order reliability method** (See UQLAB User Manual – Structural Reliability, Section 1.4.1 for further details)
- **Second-order reliability method** (See UQLAB User Manual – Structural Reliability, Section 1.4.2 for further details)

Simulation methods

Simulation methods are based on sampling the joint distribution of the random variables $\boldsymbol{W}$ and using a sample-based estimates of the integral in Eq. (1.2). At the cost of being computationally expensive, they generally have a well-characterized convergence behavior. Three sample-based algorithms are available in UQLAB:

- **Monte Carlo Simulation** (See UQLAB User Manual – Structural Reliability, Section 1.5.1 for further details)
- **Importance sampling** (See UQLAB User Manual – Structural Reliability, Section 1.5.2 for further details)
- **Subset simulation** (See UQLAB User Manual – Structural Reliability, Section 1.5.3 for further details)

1.3.1.2 Reliability index approach

The *reliability index approach* (RIA) is a classical RBDO approach based on the formulation in Eq. (1.3) which specifically uses FORM in the inner loop.
In the FORM setting, the reliability index geometrically corresponds to the distance of the origin point in the standard space $\|\boldsymbol{u}_k^*\|$, *i.e.* $\beta_k = \|\boldsymbol{u}_k^*\|$. Therefore, the inner loop then boils down to searching the design point, which can be achieved as follows:

$$\boldsymbol{u}_k^* = \underset{\boldsymbol{u}\in\mathbb{R}^M}{\operatorname{argmin}} \left\{ \|\boldsymbol{u}\|, G_k(\boldsymbol{u}) \leq 0 \right\}. \tag{1.4}$$

where $G_k(\boldsymbol{u}) = g_k(\mathcal{T}^{-1}(\boldsymbol{u}))$. This equation is usually solved by means of specialized optimization algorithms such as the Hasofer-Lind-Rackwitz-Fiessler (HLRF) algorithm and its improved version (iHLRF) (Rackwitz and Fiessler, 1978). These two methods are implemented in UQLAB, the user can find details in the UQLAB User Manual – Structural Reliability, Section 1.4.1.2. The main drawback of this approach is its low numerical efficiency despite being easy to implement.

1.3.1.3 Performance measure approach

The *performance measure approach* (PMA) is another popular RBDO solution technique. It is a well-known alternative to RIA and it also consists of two loops. In this case however, the inner loop consists of an *inverse FORM* analysis. This means that one rather sets a target reliability index and searches for the so-called *minimum performance target point* (MPTP). The associated formulation of the RBDO problem then reads:

$$\boldsymbol{d}^* = \underset{\boldsymbol{d}\in\mathbb{D}}{\arg\min}\, \mathfrak{c}\left(\boldsymbol{d}\right) \quad \text{subject to:} \quad \begin{cases} \mathfrak{f}_j\left(\boldsymbol{d}\right) \leq 0, & \{j = 1, \ldots, s\}, \\ g_k\left(\boldsymbol{x}_{\text{MPTP}_k}\left(\boldsymbol{d}\right), \boldsymbol{z}_{\text{MPTP}_k}\right) \geq 0, & \{k = 1, \ldots, n\}, \end{cases} \tag{1.5}$$

where $\boldsymbol{w}_{\text{MPTP}_k} = \{\boldsymbol{x}_{\text{MPTP}_k}, \boldsymbol{z}_{\text{MPTP}_k}\}$ is the k-th minimum performance target point associated to the design $\boldsymbol{d}$. Its search can be cast as follows in the standard Gaussian space:

$$\boldsymbol{u}_{\text{MPTP}_k} = \underset{\boldsymbol{u}\in\mathbb{R}^M}{\operatorname{argmin}} \left\{ G_k(\boldsymbol{u}), \|\boldsymbol{u}\| = \bar{\beta}_k \right\}. \tag{1.6}$$

The corresponding MPTP in the physical space can then be retrieved by an appropriate mapping $\boldsymbol{w}_{\text{MPTP}_k} = \mathcal{T}^{-1}(\boldsymbol{u}_{\text{MPTP}_k})$. This optimization problem consists of a possibly complex objective function associated to a simple spherical constraint. It is therefore easier to solve than Eq. (1.4) for RIA where the complex function is the constraint (Chateauneuf and Aoues, 2008). It can be solved with a general-purpose optimization algorithm, even though convergence may be difficult in some cases. Three main procedures are used for the efficient solution of this problem (Youn et al., 2005; Cho and Lee, 2011), namely:

- Advanced mean value (AMV), which reads:

$$\begin{aligned} &\boldsymbol{u}_{k,\text{AMV}}^{(i+1)} = \bar{\beta}_k \boldsymbol{n}\left(\boldsymbol{u}_{k,\text{AMV}}^{(i)}\right), \\ \text{where} \quad &\boldsymbol{n}\left(\boldsymbol{u}_{k,\text{AMV}}^{(i)}\right) = \frac{\nabla_{\boldsymbol{U}} G_k\left(\boldsymbol{u}_{k,\text{AMV}}^{(i)}\right)}{\|\nabla_{\boldsymbol{U}} G_k\left(\boldsymbol{u}_{k,\text{AMV}}^{(i)}\right)\|} \quad \text{and} \quad \boldsymbol{u}_{k,\text{AMV}}^{(0)} = \boldsymbol{0}. \end{aligned} \tag{1.7}$$

- Conjugate mean value (CMV), which reads:

$$\begin{aligned} &\boldsymbol{u}_{k,\text{CMV}}^{(0)} = \boldsymbol{0}, \quad \boldsymbol{u}_{k,\text{CMV}}^{(1)} = \boldsymbol{u}_{k,\text{AMV}}^{(1)}, \quad \boldsymbol{u}_{k,\text{CMV}}^{(2)} = \boldsymbol{u}_{k,\text{AMV}}^{(2)} \\ \text{and} \quad &\boldsymbol{u}_{k,\text{CMV}}^{(i+1)} = \bar{\beta}_k \frac{\boldsymbol{n}(\boldsymbol{u}_{k,\text{CMV}}^{(i)}) + \boldsymbol{n}(\boldsymbol{u}_{k,\text{CMV}}^{(i-1)}) + \boldsymbol{n}(\boldsymbol{u}_{k,\text{CMV}}^{(i-2)})}{\|\boldsymbol{n}(\boldsymbol{u}_{k,\text{CMV}}^{(i)}) + \boldsymbol{n}(\boldsymbol{u}_{k,\text{CMV}}^{(i-1)}) + \boldsymbol{n}(\boldsymbol{u}_{k,\text{CMV}}^{(i-2)})\|}, \quad \text{for } i \geq 2 \\ \text{where} \quad &\boldsymbol{n}(\boldsymbol{u}_{k,\text{CMV}}^{(i)}) = \frac{\nabla_{\boldsymbol{U}} G_k\left(\boldsymbol{u}_{k,\text{CMV}}^{(i)}\right)}{\|\nabla_{\boldsymbol{U}} G_k\left(\boldsymbol{u}_{k,\text{CMV}}^{(i)}\right)\|}. \end{aligned} \tag{1.8}$$

- Hybrid mean value (HMV), which is a combination of AMV and CMV. The method starts from the premise that AMV is efficient for convex performance functions but often shows slow convergence or sometimes even divergence when the performance function is concave. On the contrary, CMV is more efficient when it comes to concave performance functions. HMV thus checks the convexity or concavity of the performance function at each iteration to select the appropriate method. This can be checked using $\text{sign}(\zeta^{(i+1)})$, where

$$\zeta^{(i+1)} = \left(\boldsymbol{n}(\boldsymbol{u}_{k,\text{HMV}}^{(i+1)}) - \boldsymbol{n}(\boldsymbol{u}_{k,\text{HMV}}^{(i)})\right) \cdot \left(\boldsymbol{n}(\boldsymbol{u}_{k,\text{HMV}}^{(i)}) - \boldsymbol{n}(\boldsymbol{u}_{k,\text{HMV}}^{(i-1)})\right).$$

  The performance function is considered convex at $\boldsymbol{u}_{k,\text{HMV}}^{(i+1)}$ if $\zeta^{(i+1)} > 0$ and concave if $\zeta^{(i+1)} \leq 0$. The algorithm therefore reads:

$$\boldsymbol{u}_{k,\text{HMV}}^{(i+1)} = \begin{cases} \boldsymbol{u}_{k,\text{AMV}}^{(i+1)} & \text{if} \quad \zeta^{(i+1)} > 0 \\ \boldsymbol{u}_{k,\text{CMV}}^{(i+1)} & \text{if} \quad \zeta^{(i+1)} \leq 0. \end{cases} \tag{1.9}$$

Two stopping criteria are considered to check the convergence of these algorithms. They read as follows:

$$\begin{aligned} &|\boldsymbol{u}_k^{(i+1)} - \boldsymbol{u}_k^{(i)}| \leq \epsilon_U, \\ &|G_k\left(\boldsymbol{u}_k^{(i+1)}\right) - G_k\left(\boldsymbol{u}_k^{(i)}\right)| \leq \epsilon_G, \end{aligned} \tag{1.10}$$

where ϵ_U and ϵ_G are thresholds to be set.

1.3.1.4 Quantile estimate approach

Another approach in solving the RBDO problem is to consider quantiles as a measure of conservatism of the solution instead of the failure probability as introduced in Moustapha et al. (2016). By noting that the failure probability can be equivalent expressed by

$$\mathbb{P}\left(g_k\left(\boldsymbol{X}\left(\boldsymbol{d}\right), \boldsymbol{Z}\right) \leq 0\right) \leq \bar{P}_{f_k} \Leftrightarrow Q_{\alpha_k}\left(\boldsymbol{d}; g_k\left(\boldsymbol{X}\left(\boldsymbol{d}\right), \boldsymbol{Z}\right)\right) \geq 0 \tag{1.11}$$

where

$$Q_{\alpha_k}\left(\boldsymbol{d}; g_k\left(\boldsymbol{X}\left(\boldsymbol{d}\right), \boldsymbol{Z}\right)\right) = \inf\left\{q \in \mathbb{R} : \mathbb{P}\left(g_k\left(\boldsymbol{X}\left(\boldsymbol{d}\right), \boldsymbol{Z}\right) \leq q\right) \geq \alpha_k\right\}, \tag{1.12}$$

the RBDO problem in Eq. (1.1) can be recast as:

$$\boldsymbol{d}^* = \underset{\boldsymbol{d}\in\mathbb{D}}{\arg\min}\, \mathfrak{c}\left(\boldsymbol{d}\right) \quad \text{subject to:} \begin{cases} \mathfrak{f}_j\left(\boldsymbol{d}\right) \leq 0, & \{j = 1, \ldots, s\}, \\ Q_{\alpha_k}\left(\boldsymbol{d}; g_k\left(\boldsymbol{X}\left(\boldsymbol{d}\right), \boldsymbol{Z}\right)\right) \geq 0, & \{k = 1, \ldots, n\}, \end{cases} \tag{1.13}$$

with $\alpha_k = \bar{P}_{f_k}$.
In practice the computation of the quantiles in Eq. (1.12) is carried out using crude Monte Carlo sampling.

1.3.2 Mono-level approach

Mono-level approaches aim at avoiding the reliability analysis at each iteration of the optimization process. Instead, the problem is converted into an equivalent single loop deterministic process. The single loop approach (SLA) which belongs to this class of methods is currently implemented in UQLAB.
In SLA, the equivalent deterministic problem is obtained by enforcing the Karush-Kuhn-Tucker optimality conditions of the reliability analysis as additional constraints (Liang et al., 2004). This reads as follows:

$$\boldsymbol{d}^* = \underset{\boldsymbol{d}\in\mathbb{D}}{\arg\min}\, \mathfrak{c}\left(\boldsymbol{d}\right) \quad \text{subject to:} \begin{cases} \mathfrak{f}_j\left(\boldsymbol{d}\right) \leq 0, & \{j = 1, \ldots, s\}, \\ g_k\left(\boldsymbol{x}_{\text{MPTP}_k}\left(\boldsymbol{d}\right), \boldsymbol{z}_{\text{MPTP}_k}\right) \geq 0, & \{k = 1, \ldots, n\}, \end{cases} \tag{1.14}$$

where

$$\begin{aligned} \boldsymbol{x}_{\text{MPTP}_k} &= \boldsymbol{\mu}_{\boldsymbol{X}} - \boldsymbol{\sigma}_{\boldsymbol{X}} \cdot \bar{\beta}_k \cdot \boldsymbol{\alpha}_k, \\ \boldsymbol{z}_{\text{MPTP}_k} &= \boldsymbol{\mu}_{\boldsymbol{Z}} - \boldsymbol{\sigma}_{\boldsymbol{Z}} \cdot \bar{\beta}_k \cdot \boldsymbol{\alpha}_k. \end{aligned} \tag{1.15}$$

In these equations, $\boldsymbol{\mu}_{\boldsymbol{X}}$ and $\boldsymbol{\mu}_{\boldsymbol{Z}}$ are the mean values of the design and environmental vari-

ables, respectively. While the former may change from one iteration to the next, the latter remain the same throughout the optimization. The standard deviations of the design and environmental variables are gathered in $\boldsymbol{\sigma} = \{\sigma_{\boldsymbol{X}}, \sigma_{\boldsymbol{Z}}\}$. Finally, $\boldsymbol{\alpha}_k$ is the normalized gradient of constraint k at the previous MPTP and reads:

$$\boldsymbol{\alpha}_k = \boldsymbol{\sigma} \cdot \nabla_{g_k}\left(\boldsymbol{x}_{\text{MPTP}_k}(\boldsymbol{d}), \boldsymbol{z}_{\text{MPTP}_k}\right) / \| \boldsymbol{\sigma} \cdot \nabla_{g_k}\left(\boldsymbol{x}_{\text{MPTP}_k}(\boldsymbol{d}), \boldsymbol{z}_{\text{MPTP}_k}\right) \|. \tag{1.16}$$

This value is updated only in iterations where the mean value of the design variables $\boldsymbol{\mu}_{\boldsymbol{X}}$ evolves.

The above formulation assumes that all the random variables are independent and normally distributed. When this is not the case, the equivalent mean and standard deviations may be found using a non-linear transformation, namely the *Rackwitz-Fiessler two-parameter equivalent normal method* (Wu et al., 2001). This consists in assuming that the CDF and PDF of the actual non-normal and the equivalent normal distributions are equal at the current MPTP, thus yielding:

$$\begin{aligned} \boldsymbol{\sigma} &= \varphi\left[\Phi^{-1}\left(F_{\boldsymbol{W}}\left(\boldsymbol{w}_{\text{MPTP}_k}\right)\right)\right] / f_{\boldsymbol{W}}\left(\boldsymbol{w}_{\text{MPTP}_k}\right), \\ \boldsymbol{\mu} &= \boldsymbol{w}_{\text{MPTP}_k} - F_{\boldsymbol{W}}\left(\boldsymbol{w}_{\text{MPTP}_k}\right) \cdot \boldsymbol{\sigma}, \end{aligned} \tag{1.17}$$

where $\boldsymbol{\mu} = \{\boldsymbol{\mu}_{\boldsymbol{X}}, \boldsymbol{\mu}_{\boldsymbol{Z}}\}$, $\boldsymbol{w}_{\text{MPTP}_k} = \{\boldsymbol{x}_{\text{MPTP}_k}, \boldsymbol{z}_{\text{MPTP}_k}\}$ and the product and division in the above equation are applied component-wise.

1.3.3 Decoupled approach

The decoupled approaches are a set of RBDO methods which consist in sequentially solving a deterministic optimization problem followed by a reliability analysis. One such technique is currently implemented in UQLAB, namely the *sequential optimization and reliability assesment* (SORA).

SORA (Du and Chen, 2004) converts the probabilistic constraint into an equivalent deterministic constraint through the use of the minimum performance target point. At each cycle, the constraint is evaluated by shifting the design variables according to the MPTP found in the reliability analysis of the previous cycle. At cycle i, the optimization problem in Eq. (1.1) is approximated by:

$$\boldsymbol{d}^* = \underset{\boldsymbol{d}\in\mathbb{D}}{\arg\min}\, \mathfrak{c}(\boldsymbol{d}) \quad \text{subject to:} \quad \begin{cases} \mathfrak{f}_j(\boldsymbol{d}) \leq 0, & \{j = 1, \ldots, s\}, \\ g_k\left(\boldsymbol{d} - \boldsymbol{s}_k^{(i)}, \boldsymbol{z}_k^{(i-1)}\right) \geq 0, & \{k = 1, \ldots, n\}, \end{cases} \tag{1.18}$$

where $\boldsymbol{s}_k^{(i)} = \boldsymbol{d}^{(i-1)} - \boldsymbol{x}_{\text{MPTP}_k}^{(i-1)}$ and $\boldsymbol{w}_{\text{MPTP}_k}^{(i-1)} = \left\{\boldsymbol{x}_{\text{MPTP}_k}^{(i-1)}, \boldsymbol{z}_{\text{MPTP}_k}^{(i-1)}\right\}$ is the MPTP of the constraint k at the cycle $i-1$. This point is found by means of an inverse FORM analysis as detailed in Section 1.3.1.3.

1.4 RBDO solution using surrogate models

Reliability-based design optimization is quite expensive in terms of model evaluations. Because of this shortcoming, applications have long been limited to toy mathematical examples or highly simplified problems. Methods based on the approximation approaches (e.g. RIA, SLA, SORA) were introduced as contributions to solve this issue. However, they may not be accurate when the limit-state surface is highly non-linear or when multiple design points exist. Moreover, real-world problems are often solved using time-consuming high-fidelity computational models (e.g. finite element). For such applications, even the approximation methods based on FORM become relatively expensive. In the past decade, surrogate modelling has been introduced as a means to approximate expensive-to-evaluate functions. They have been successfully applied to reliability analysis and reliability-based design optimization, thus allowing for the solution of complex industrial problems. UQLAB offers a set of metamodelling techniques which can be used for reliability analysis and optimization. The basic idea is to simply replace the limit-state functions introduced in the previous equations by an equivalent metamodel-based approximation which has been built beforehand. In its current stage, UQLAB offers the following metamodeling approaches:

- 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);
- Low-rank approximation (UQLAB User Manual – Canonical low-rank approximations);
- Support vector machines (UQLAB User Manual – Support vector machines regression).

The implementation of the surrogate-assisted solution schemes follows the unified framework introduced in Moustapha and Sudret (2019). Some important concepts are reminded in the following paragraphs.

1.4.1 Augmented space for RBDO

In the context of RBDO, metamodels are used for the approximation of the limit-state functions. Their interest is prominent in two-level approaches where the inner loop requires a reliability analysis often carried out by simulation. An easy way to introduce metamodeling in this framework is to build a local surrogate model at each iteration for each reliability analysis. This is however not an optimal approach and may become quite expensive when the optimization requires many iterations to converge. A more efficient approach consists in building a unique and global surrogate model in the so-called augmented space (Dubourg et al., 2011; Taflanidis and Beck, 2008). The basic idea behind this concept is to build a space

which accounts for both design and environmental variables at once. Further this space defines a confidence interval where the metamodels are built to avoid extrapolation when evaluating them. The implementation in UQLAB follows the development in Moustapha et al. (2016). This augmented space results from the following tensor product $\mathbb{X} \times \mathbb{Z}$ defined by:

$$\mathbb{X} = \prod_{i=1}^{M_d} \left[q_{d_i}^-, q_{d_i}^+ \right], \tag{1.19}$$

with

$$q_{d_i}^- = F_{X_i|d_i^-}^{-1}\left(\alpha_{d_i}/2\right) \quad \text{and} \quad q_{d_i}^+ = F_{X_i|d_i^+}^{-1}\left(1-\alpha_{d_i}/2\right), \tag{1.20}$$

where $F_{X_i|d_i^-}^{-1}$ and $F_{X_i|d_i^+}^{-1}$ are the inverse cumulative distribution functions of the design random variables X_i at the extreme lower and upper bounds of the design space d_i^- and d_i^+ and α_{d_i} is the probability of sampling outside the augmented space in the i-th dimension.
For the environmental variables, two cases can be considered:

- The first approach consists in a similar construction using a hypercube as for the design variables. This can be set up as follows:

$$\mathbb{Z} = \prod_{j=1}^{M_z} \left[q_{z_j}^-, q_{z_j}^+ \right], \tag{1.21}$$

  where

$$q_{z_j}^- = F_{Z_j}^{-1}\left(\alpha_{z_j}/2\right) \quad \text{and} \quad q_{z_j}^+ = F_{Z_j}^{-1}\left(1-\alpha_{z_j}/2\right), \tag{1.22}$$

  In these equations, α_{z_j} is the probability of sampling outside the augmented space in the j-th dimension for the environmental variables and $F_{Z_i}^{-1}$ is the inverse cumulative distribution function of Z_i. This approach is referred to as *hypercube* approach here as the resulting augmented space is a hypercube.

- Alternatively, one may sample directly the environmental variables according to their distributions as they do not evolve during optimization. This approach is referred to as *hybrid* in UQLAB.

Note: The typical values of α_{d_i} and α_{z_i} are extremely small probabilities, *e.g.* 10^{-4}. In UQLAB, they are set by default to a relatively large value of 10^{-2} because it is assumed that active learning will allow sampling points further in the tails of the input parameters distribution if needed.

1.4.2 Adaptive surrogate modelling

In high dimensional problems, bulk surrogate modelling on a unique space-filling experimental design is not often enough as the number of samples needed to cover the space in a satisfactory manner may be too large. Numerous approaches have been developed that rather build the surrogate model by iteratively enriching an initial experimental design of

small size. The basic idea is to add samples that are most likely to improve the accuracy of the metamodel in the region of interest which, for reliability, is the vicinity of the limit-state surface (Echard et al., 2011; Bect et al., 2012). These methods consist in defining a so-called *learning function* whose evaluation indicates the sample points whose addition in the current experimental design is likely to improve the most the metamodel accuracy. The generic procedure is as follows:

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 responses $\mathcal{Y} = \left\{y^{(1)}, \ldots, y^{(N_0)}\right\} = \left\{g\left(\boldsymbol{x}^{(1)}\right), \ldots, g\left(\boldsymbol{x}^{(N_0)}\right)\right\}$;
2. Train a metamodel $\widehat{g}$ based on the experimental design $\{\mathcal{X}, \mathcal{Y}\}$;
3. Generate a large set of N_{MC} candidate samples $\mathcal{S} = \left\{\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(N_{MC})}\right\}$;
4. Choose the best or a set of next samples $\boldsymbol{s}^*$ to be added to the experimental design $\mathcal{X}$ based on an appropriate learning function;
5. Check whether some convergence criteria are met. If they are, skip to Step 7, otherwise continue with Step 6;
6. Add $\boldsymbol{s}^*$ and the corresponding limit-state response $y^* = g(\boldsymbol{s}^*)$ to the experimental design. Return to Step 2.
7. End algorithm.

Various enrichment strategies are implemented in UQLAB as described in the sequel.

1.4.3 Learning functions

A wide variety of learning functions exist in the literature. They are most of the time metamodel-dependent, *i.e.* a learning function is developed for a specific type of metamodel. In UQLAB, four learning functions are implemented. The following table shows the correspondence with the available metamodels.

Table 1: Available learning functions and metamodels for which they apply.

Learning function	Kriging	PC-Kriging	PCE	LRA	SVR
Expected feasibility function (EFF)	✓	✓	✗	✗	✗
Deviation number (U)	✓	✓	✗	✗	✗
Constrained min-max (CMM)	✓	✓	✓	✓	✓
Fraction of bootstrap replicates (FBR)	✗	✗	✓	✗	✗

1.4.3.1 Deviation number

The deviation number or U-function is a learning function developed by Echard et al. (2011) whose idea is to estimate the probability of misclassifying a point in a reliability analysis when

the original model is replaced by a Gaussian process approximation. The function reads:

$$U(\boldsymbol{x}) = \frac{\left|\mu_{\widehat{g}}(\boldsymbol{x})\right|}{\sigma_{\widehat{g}}(\boldsymbol{x})}, \tag{1.23}$$

where $\mu_{\widehat{g}}(\boldsymbol{x})$ and $\sigma_{\widehat{g}}(\boldsymbol{x})$ are the prediction mean and standard deviation of $\widehat{g}$.
Small values of U correspond to points that are either close to the limit-state surface or that belong to areas of the space with large Kriging variance (due to scarcity of data). Following this information, the next best point to add in order to refine the limit-state surface is given by:

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\mathcal{S}}{\arg\min}\, U(\boldsymbol{s}). \tag{1.24}$$

1.4.3.2 Expected feasibility function

The expected feasibility function is another popular learning function developed by Bichon et al. (2008) in the context of active Kriging reliability analysis and reads:

$$\begin{aligned} EFF(\boldsymbol{x}) = \mu_{\widehat{g}}(\boldsymbol{x}) &\left[2\Phi\left(\frac{-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \Phi\left(\frac{-\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \Phi\left(\frac{\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right)\right] \\ - \sigma_{\widehat{g}}(\boldsymbol{x}) &\left[2\varphi\left(\frac{-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \varphi\left(\frac{-\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \varphi\left(\frac{\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right)\right] \\ &+ \epsilon\left[\Phi\left(\frac{\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right) - \Phi\left(\frac{-\epsilon-\mu_{\widehat{g}}(\boldsymbol{x})}{\sigma_{\widehat{g}}(\boldsymbol{x})}\right)\right], \end{aligned} \tag{1.25}$$

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

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\mathcal{S}}{\arg\max}\, EFF(\boldsymbol{s}). \tag{1.26}$$

1.4.3.3 Constrained min-max

This approach is an adaptation of the learning function proposed by Basudhar and Missoum (2008) in the context of structural reliability using support vector machines. They suggest a constrained min-max optimization problem: the next enrichment point is the one lying on the limit-state surface while being the furthest from all training points. Here we consider a discretized version of this optimization problem, *i.e.* the next point is chosen from the sample set $\mathcal{S}$ in two steps. The first one is to select a small subset of the samples that are the closest to the limit-state surface, say $1-10\%$. The next best point is then the one from this subset that is the furthest away from the existing training points. Mathematically, this reads:

$$\boldsymbol{s}^* = \max_{\boldsymbol{s}\in\mathcal{S}'} \min_{i=1,\dots,N} \|\boldsymbol{s} - \boldsymbol{x}^{(i)}\| \tag{1.27}$$

where $\left\{\boldsymbol{x}^{(i)}, i = 1,, \dots, N\right\}$ are the current training points and

$$\mathcal{S}' = \left\{\boldsymbol{s} \in \mathcal{S} : |g(\boldsymbol{s})| < q\right\}. \tag{1.28}$$

Here q is a α-quantile of $|g(\boldsymbol{s})|$. For instance $\alpha = 0.05$ corresponds to taking a subset corresponding to the 5% closest points to the limit-state surface.

1.4.3.4 Fraction of bootstrap replicates

This learning function was proposed by Marelli and Sudret (2018) in the context of reliability analysis using polynomial chaos expansions. Details on bootstrap-PCE can be found in UQLAB User Manual – Polynomial Chaos Expansions. In a nutshell, the approach first consists in building B PCE models using B bootstrap replicates of the experimental design. The fraction of failed bootstrap replicates is then defined as:

$$U_{FBR}\left(\boldsymbol{s}\right) = \left|\frac{B_s\left(\boldsymbol{s}\right) - B_f\left(\boldsymbol{s}\right)}{B}\right|, \tag{1.29}$$

where $B_s\left(\boldsymbol{s}\right)$ and $B_f\left(\boldsymbol{s}\right) \in \{0, \ldots, B\}$ are the number of safe and failed PC-bootstrap replicates at the point $\boldsymbol{s}$, respectively. When the points are consistently described as either safe or failed with respect to all bootstrap replicates, U_{FBR} tends to 1. On the contrary, when the sign of the estimate varies a lot within replications, U_{FBR} tends to 0. The latter case corresponds to points in the space that are not well estimated by the PCE model and/or are likely to be located near the limit-state surface. Using this learning function, the next best point to add is defined by:

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\mathcal{S}}{\arg\min}\, U_{FBR}(\boldsymbol{s}). \tag{1.30}$$

1.4.4 Convergence criteria

A convergence criterion is used to stop the enrichment of the experimental design when the limit-state surface is accurate enough for further use in reliability analysis and optimization. Three convergence criteria are available in UQLAB:

- **On the learning function**: The algorithm stops when the learning function reaches a given threshold.

  For the U-function, this reads:

  $$\min_{\boldsymbol{s}\in\mathcal{S}} U\left(\boldsymbol{s}\right) > 2. \tag{1.31}$$

  As for the expected feasibility function, the stopping criterion reads:

  $$\max_{\boldsymbol{s}\in\mathcal{S}} EFF\left(\boldsymbol{s}\right) \geq 10^{-3}. \tag{1.32}$$

  These two criteria were introduced in the original contributions on the techniques (Echard et al., 2011; Bichon et al., 2008). However they have been shown to be extremely conservative.

- **On the size of the failure domain**: the algorithm stops when the size of the failure domain in the augmented space does not change considerably within iterations. The

criterion reads:

$$\frac{\left|N_f^{(i-1)} - N_f^{(i)}\right|}{N_f^{(i-1)}} < \epsilon_{D_f}, \tag{1.33}$$

where $N_f^{(i)}$ is the number of failed samples, *i.e.* the size of the subset $\{\boldsymbol{s} \in \mathcal{S} : \widehat{g}(\boldsymbol{s}) < 0\}$ at the i-th enrichment iteration. For robustness, convergence is assumed only when this criterion is respected in two iterations in a row.

- **On the stability of the fail/safe predictions**: The algorithm stops when the number of points whose prediction sign goes from safe to unsafe or vice-versa within two iterations of enrichment is small enough. This criterion reads:

$$\frac{\text{Card}\left(\left\{\boldsymbol{s} \in \mathcal{S} : \widehat{g}^{(i)}(\boldsymbol{s}) \times \widehat{g}^{(i-1)}(\boldsymbol{s}) < 0\right\}\right)}{N_s} < \epsilon_{SC}, \tag{1.34}$$

where Card denotes the cardinality of a set and $\widehat{g}^{(i)}(\boldsymbol{s})$ is the metamodel prediction at the point $\boldsymbol{s}$ at the i-th enrichment iteration. For robustness, convergence is assumed only when this criterion is respected in two iterations in a row.

1.5 Optimization algorithms

The solution of the different RBDO formulations introduced above can be achieved by general-purpose optimization algorithms. UQLAB offers both local and global algorithms. The former rely on local information, often first- and/or second-order derivatives of the objective and constraint functions, to move to the next iterate. By their very nature, they are likely to converge to a local solution which may differ from the global one, but within a limited number of iterations and therefore reduced computational cost. On the contrary, global algorithms explore the entire design domain. They are then more likely to find the global solution even though this may come at a higher cost. The following algorithms are currently offered:

- Interior-point (see `fmincon` in MATLAB);
- Sequential quadratic programming (see `fmincon` in MATLAB);
- Genetic algorithm (see `ga` in MATLAB);
- Constrained $(1+1)$-CMA-ES (Arnold and Hansen (2012), See UQLIB user manual);
- Hybrid algorithms, which refine the solution identified by a genetic algorithm or $(1+1)$-CMA-ES by an additional gradient-based minimization.

1.5.1 Numerical considerations

In general, the RBDO module of UQLAB is designed such that general-purpose algorithms can be used to solve the problem. However some precautions need to be taken:

- *Common random numbers*: To be able to use general-purpose optimization algorithms with simulation approaches, especially those relying on gradient information, this module uses the concept of common random numbers (Spall, 2003; Taflanidis and Beck, 2008). This consists in using the same stream of random numbers to generate samples within different iterations of the optimization process. This smooths out the problem but also introduces a bias which can be mitigated by increasing the size of the sample set or by multiple runs of the solution. Common random numbers cannot be used efficiently with subset simulation and importance sampling. Therefore, it is advised to only consider global optimization algorithms when using advanced simulation methods.
- Finite difference step size: When using simulation methods, a too small finite difference step size may cause the optimization algorithm to diverge. In some cases, it is necessary to increase the default step size and/or reduce the variance of the estimated failure probability by increasing the sample size.

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: Column under compression

The benchmark of choice to showcase the methods described in Section 1.3 is a RBDO problem related to a column under compression. We aim at minimizing the cross-sectional area of a column of rectangular cross-section $b \times h$ submitted to a service load F_{ser}, while avoiding buckling. Buckling occurs when the service load is larger than the critical Euler force:

$$F_{\text{cr}} = \frac{\pi^2 EI}{L^2}, \tag{2.1}$$

where L is the column length, E is its constitutive material Young's modulus and $I = bh^3/12\,(b > h)$ is the column smallest principal moment of inertia.
The RBDO problem therefore reads:

$$\boldsymbol{d}^* = \underset{\boldsymbol{d}\in[150,\,350]^2}{\arg\min}\ \mathfrak{c}\left(\boldsymbol{d}\right) = bh \quad \text{subject to:} \quad \begin{cases} h - b \leq 0, \\ \mathbb{P}\left(g\left(\boldsymbol{d}, \boldsymbol{Z}\right) = F_{\text{ser}} - k\frac{\pi^2 Ebh^3}{12L^2}\right) \leq \bar{P}_f, \end{cases} \tag{2.2}$$

where k is a factor that accounts for random noise affecting the Euler force and $\bar{P}_f = 0.05$ is the target failure probability.
The probabilistic model for this example, presented in Table 2, consists of three environmental variables: $\boldsymbol{z} = \{k,\, E,\, L\}$. The design parameters b and h and the service load $F_{\text{ser}} = 1.4622 \times 10^6$ are considered deterministic.

Table 2: Probabilistic modelling of the parameters describing the environmental variables.

Name	Distributions	Mean	C.O.V (%)
k	Lognormal	0.6	10
E	Lognormal	$10,000$	5
L	Lognormal	$3,000$	1

The exact solution is analytically available (Moustapha, 2016): $b^* = h^* = 238.45$ mm.
A UQLAB example script that showcases how to deploy all of the algorithms available in the RBDO module on this problem can be found in the example file
`Examples/RBDO/uq_Examples_RBDO_01_ColumnCompression.m`.

2.2 Problem set-up

The ingredients needed for solving a reliability-based design optimization problem are (Section 1.2):

- a set of hard constraints/limit-state functions ;
- a probabilistic model of the random vector $\boldsymbol{Z}$, if any;
- the following information that define the optimization problem:
  - the cost function;
  - the design space;
  - the target failure probability.

The UQLAB framework is first initialized with the following command:

```
uqlab
```

The limit-state fucntion $g(\boldsymbol{d}, \boldsymbol{Z})$ in Eq. (2.2) is specified in UQLAB as a MODEL object. In this case, we can directly use a vectorized implementation of the g-function in Eq. (2.2):

```
MOpts.mFile = 'uq_columncompression_constraint';
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 lognormal variables as specified in Table 2 can be created as follows:

```
Iopts.Marginals(1).Name = 'k' ;
Iopts.Marginals(1).Type = 'Lognormal' ;
Iopts.Marginals(1).Moments = [0.6 0.1*0.6] ;
Iopts.Marginals(2).Name = 'E' ;
Iopts.Marginals(2).Type = 'Lognormal' ;
Iopts.Marginals(2).Moments = [1e4 0.05*1e4] ;
Iopts.Marginals(3).Name = 'L' ;
Iopts.Marginals(3).Type = 'Lognormal' ;
Iopts.Marginals(3).Moments = [3e3 0.01*3e3] ;

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.

Once these two objects are defined, the RBDO problem can be setup as follows:

```
RBDopt.Type = 'RBDO' ;
% Cost function
RBDopt.Cost.mString = 'X(:,1) .* X(:,2)' ;
% Hard constraint
RBDopt.LimitState.Model = myModel ;
% Soft constraint
RBDopt.SoftConstraints.mString = 'X(:,2) - X(:,1)' ;
% Design variables (both deterministic parameters)
RBDopt.Input.DesVar(1).Name = 'b' ;
RBDopt.Input.DesVar(1).Type = 'constant' ;
RBDopt.Input.DesVar(2).Name = 'h' ;
RBDopt.Input.DesVar(2).Type = 'constant' ;
% Environmental variables
RBDopt.Input.EnvVar = myInput ;
% Design space
RBDopt.Optim.Bounds = [150 150; 350 350];
% Target failure probability
RBDopt.TargetPf = 0.05 ;

% Run the analysis and add it to UQLab
myRBDO = uq_createAnalysis(RBDopt) ;
```

Note: When not specified by the user, the following default values are used:

- RBDO method: Generalized two-level
- Reliability analysis: Monte Carlo simulation with 10^4 sample points using common random numbers
- Optimization algorithm: Constrained $(1+1)$-CMA-ES
- Starting point: Center of the design space

Once the analysis is performed, a report with the results can be printed with the `uq_print` command:

```
uq_print(myRBDO);

%------------------------------ RBDO ------------------------------%

%------------------ Optimal cost
--------------
| Fstar      |
--------------
| 5.6875e+04 |
--------------

%------------------ Optimal design
----------------------------
| b          | h          |
----------------------------
| 2.3848e+02 | 2.3848e+02 |
----------------------------

%------------------ Constraints at solution
```

```
--------------
| Pf_1       |
--------------
| 5.0000e-02 |
--------------

%------------------ Method
RBDO Method:         Two-level with Monte Carlo simulation

%------------------ Optimization
Algorithm:           Constrained (1+1)-CMA-ES
Iterations:          500
Model evaluations:   5000000
Exit message:        Maximum number of generations reached

%-----------------------------------------------------------------%
```

The results can be visualized graphically using the `uq_display` command. In general, when using `uq_display`, two or three plots are displayed that show the evolution of the cost and constraints throughout the optimization process. For the latter, the quantity displayed depends on the selected RBDO method. For the generalized two-level approach and RIA it corresponds to either the failure probability or the reliability index (according to the selected type of constraints). For SORA, SLA and PMA, it corresponds to the limit-state function evaluated at the most probable target point (MPTP) or its approximation. Finally for Quantile-based RBDO (QMC), the corresponding quantity is the estimated quantile at each iteration.

```
uq_display(myRBDO) ;
```

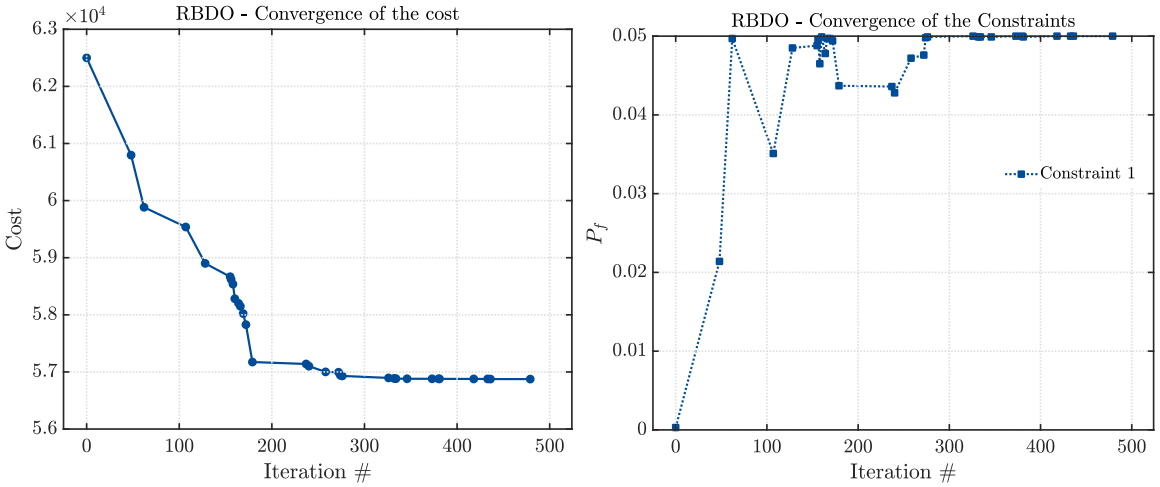

Figure 1: Graphical visualization of the convergence of the RBDO algorithm.

Note: The $(1+1)$-CMA-ES is a random search algorithm. As such, the actual convergence curve is very noisy. For clarity, only the sampled points that are feasible and actually improve the current best design are shown.

Note: When using GA or HGA, the history of the constraints is not displayed

For problems with dimension higher than two, a representation of the final design with respect to the bounds of the design space is also displayed. Figure 2 shows the third display for the bracket structure example in `uq_Example_RBDO_02_BracketStructure.m`, where the design space is three-dimensional. The figure shows a polygon, each vertex of which represents the upper bound of a design variable. The center of the polygon represents the lower bounds. The optimal design is then represented by the green diamonds placed on the segments that connect the center and the vertex. Their relative positions represent their distances to the physical bounds of the variables.

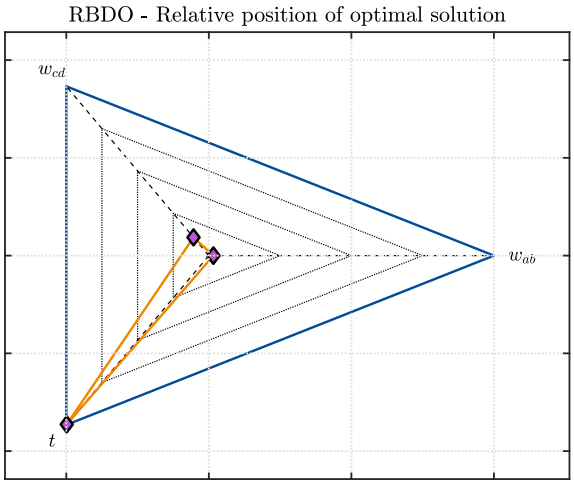

Figure 2: Graphical visualization of the optimal design for problems with dimension larger than 2.

2.3 Accessing the results

The optimization results can be accessed in the `myRBDO.Results` structure:

```
myRBDO.Results
ans =
struct with fields:

             dstar: [238.5009 238.4779]
             Fstar: 5.6877e+04
           exitMsg: 'Maximum number of generations reached'
            output: [1x1 struct]
  ModelEvaluations: 5000000
           History: [1x1 struct]
```

In these results, `dstar` is the solution of the RBDO problem and `Fstar` is the corresponding cost. `exitMsg` is a message that describes the termination condition of the optimization algorithm. The `output` structure gives detailed information on the optimization process such as the number of iterations, cost function evaluations or the exit flag and message of the optimization algorithm. For an exact description of the fields of the `output` structure, the user may refer to the documentation of the optimization algorithm (directly in MATLAB or in UQLIB user manual). `ModelEvaluations` is the number of evaluations of the computational or surrogate model throughout the optimization.
Finally, the recorded history of the optimization process is stored in the structure `History`, for instance here:

```
myRBDO.Results.History
ans =
struct with fields:

            X: [500x2 double]
         Cost: [500x1 double]
  Constraints: [500x1 double]
```

The fields of the `Results.History` structure showcase the evolution of the design variables (`X`), the cost (`Cost`) and the constraints (`Constraints`) throughout the iterations. The value given by the latter depends on the RBDO formulation. For the generalized two-level approach and RIA this value corresponds to either the failure probability or the reliability index (according to the selected type of constraints). For SORA, SLA and PMA, this corresponds to the limit-state function value at the most probable target point (MPTP) or its approximation. Finally for QMC, the corresponding value is the estimated quantile at each iteration.

2.4 Specifying different methods

2.4.1 Generalized double-loop approach

To use a double loop method, the following option should be set:

```
RBDopt.Method = 'Two-Level' ;
```

By default, the reliability analysis in the inner loop is crude Monte Carlo sampling. In general, provided that the user takes some precautions (see Section 1.5.1), any method from the

structural reliability module can be used. For instance, the following line of code specifies subset simulation (Au and Beck, 2001) for the inner loop.

```
RBDopt.Reliability.Method = 'Subset' ;
```

Depending on the chosen method, different field options that can be set. The specifications for the selected reliability analysis can be set in the field `.Reliability` (See UQLAB User Manual – Structural Reliability for an extensive guide on all the available features of the reliability module). For example, the following lines of code set the intermediate predefined failure probability and the sample batch size for subset simulation:

```
RBDopt.Reliability.SubsetSim.p0 = 0.2 ;
RBDopt.Reliability.Simulation.BatchSize = 1000 ;
```

Note: When using Monte Carlo simulation, the user can set only one of the options `.Simulation.BatchSize` or `.Simulation.MaxSampleSize` (as opposed to the reliability module, where the two options can be set independently). This allows UQLAB to enforce the use of the same sample size throughout the optimization process. Likewise, the option `.Simulation.TargetCoV` is ignored if set by the user.

2.4.2 Reliability index approach (RIA)

The reliability index approach can be set using the following command:

```
RBDopt.Method = 'RIA' ;
```

RIA actually corresponds to a two-level approach with FORM in the inner loop and the use of the reliability index as constraint instead of the failure probability (See Eq. (1.3)). Specific options for FORM can be specified using the field `.Reliability.FORM`, *e.g.*:

```
RBDopt.Reliability.FORM.StopU = 1e-3;
RBDopt.Reliability.FORM.MaxIterations = 100;
```

Note: Gradients computation options when using HLRF or iHLRF in FORM can be set using the field `.Reliability.Gradient` (See also UQLAB User Manual – Structural Reliability).

2.4.3 Performance measure approach (PMA)

The performance measure approach can be set using the following command:

```
RBDopt.Method = 'PMA' ;
```

PMA actually corresponds to a two-level approach with an inverse FORM in the inner loop and the use of the reliability index as constraint instead of the failure probability (See Eq. (1.3)). Specific options for inverse FORM can be specified using the field `.Reliability.invFORM`, *e.g.*:

```
RBDopt.Reliability.invFORM.Algorithm = 'AMV';
RBDopt.Reliability.invFORM.stopG = 1e-2;
```

Note: Gradient computation options when using AMV, CMV or HMV algorithms in inverse FORM can be set using the field `.Reliability.Gradient` (See also UQLAB User Manual – Structural Reliability).

2.4.4 Quantile-based formulation (QMC)

The quantile-based formulation (Eq. (1.13)) can be set using the following command:

```
RBDopt.Method = 'QMC' ;
```

This approach is a double-loop where the inner loop consists in estimating a quantile using Monte Carlo simulation. Specific options of Monte Carlo can be set using the field `.Reliability.Simulation`, *e.g.* to set the maximum sample size and the sampling method:

```
RBDopt.Reliability.Simulation.MaxSampleSize = 1e5 ;
RBDopt.Reliability.Simulation.Sampling = 'MC' ;
```

Note: The value of the quantile level α in Eq. (1.13) is implicitly calculated given the target failure probability or reliability index and the limit-state options (field `.LimitState.CompOp`, see Section 2.7.4).

2.4.5 Sequential optimization and reliability assessment (SORA)

SORA can be set using the following command:

```
RBDopt.Method = 'SORA' ;
```

Since SORA is the only decoupled method implemented in UQLAB, setting the RBDO method to `'decoupled'` automatically selects SORA. In this case, any specified reliability method is ignored.
When using SORA, the command `uq_display` displays Figure 3 below.
Note that the horizontal label shows the cycles of SORA, each cycle corresponding to full deterministic optimization (see Section 1.3.3). The vertical label of the right panel displays the resulting limit-state function at the end of each SORA cycle.

2.4.6 Single-loop approach (SLA)

The single-loop approach can be set using the following command:

```
RBDopt.Method = 'SLA' ;
```

Since SLA is the only mono-level approach implemented in UQLAB, setting the RBDO method to `'Mono-Level'` automatically selects SLA. In this case, any specified reliability method is ignored.

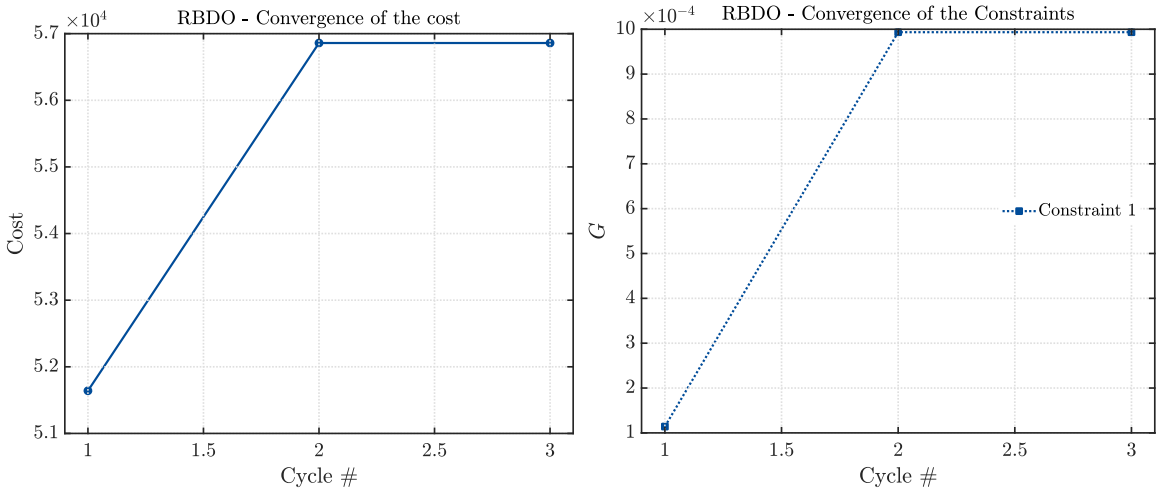

Figure 3: Graphical visualization of the convergence of the RBDO algorithm when using SORA.

2.5 Surrogate-based RBDO

2.5.1 Specifying a surrogate model

When the provided constraint model is expensive-to-evaluate, the user can choose to replace it by a metamodel that is built automatically by UQLAB. To this end, the user only needs to provide a metamodel type and all necessary options to define that metamodel. To specify a Kriging model for instance, the field `.Metamodel.Type` should be set as:

```
RBDopt.Metamodel.Type = 'Kriging' ;
```

The specific options necessary to build the metamodel depend on the metamodel type. For instance, one can fully define a Kriging model as follows:

```
% Choose Kriging as metamodel type
RBDopt.Metamodel.Type = 'Kriging'
% Set Kriging options
RBDopt.Metamodel.Kriging.ExpDesign.Sampling = 'LHS'
RBDopt.Metamodel.Kriging.ExpDesign.NSamples = 30
```

In this example, a Kriging model based on a Latin Hypercube experimental design of size 30 is used. The INPUT object necessary to sample the ED is defined internally based on the augmented space as explained in Section 1.4.1.

In the above example, only the mandatory options were set. In general, the user can set all options available for the selected metamodel type. For instance, to set the correlation function of the Kriging model to Gaussian, the following code can be used:

```
RBDopt.Metamodel.Kriging.Corr.Family = 'Gaussian';
```

For an exhaustive list of the metamodels options, the user is referred to the corresponding metamodel's UQLAB User Manual:

- For `.Metamodel.Kriging`, see UQLAB User Manual – Kriging (Gaussian process modelling);
- For `.Metamodel.PCE`, see: UQLAB User Manual – Polynomial Chaos Expansions;
- For `.Metamodel.PCK`, see UQLAB User Manual – PC-Kriging;
- For `.Metamodel.LRA`, see UQLAB User Manual – Canonical low-rank approximations;
- For `.Metamodel.SVR`, see UQLAB User Manual – Support vector machines regression.

2.5.2 Using adaptive surrogate modelling

UQLAB offers the possibility to adaptively build the metamodel (also known as *active learning*) for improved efficiency (see Section 1.4.2). To enable active learning, the user simply needs to select one of the `.Metamodel.Enrichment` options. All the options related to the enrichment process can be set using the `.Metamodel.Enrichment` field. For instance, to set the learning function and the size of the candidate set for enrichment (See Section 1.4.2), the following code can be used:

```
RBDopt.Metamodel.Enrichment.LearningFunction = 'CMM';
RBDopt.Metamodel.Enrichment.SampleSize = 1e4 ;
```

An exhaustive list of all options available for active learning is given in Table 14.

Note: Not all learning functions can be used with any metamodel. Table 1 on page 10 shows the allowed combinations. Likewise, some of the convergence criteria are available only for specific learning functions.

2.5.3 Defining the augmented space

The augmented space can be built using two different schemes for the environmental variables, which can be set with the option `.AugSpace.Method`. This option is set by default to `'hypercube'`, *i.e.* the environmental variables are sampled within a hypercube whose bounds are defined following Section 1.4.1. Alternatively, the environmental variables can be sampled directly from their respective distributions if the following code is used (See Section 1.4.1):

```
RBDopt.AugSpace.Method = 'hybrid' ;
```

Further, the probability of sampling outside the augmented space for the design and environmental variables can be set, say to 10^{-3}, using the options:

```
RBDopt.AugSpace.DesAlpha = 1e-3 ;
RBDopt.AugSpace.EnvAlpha = 1e-3;
```

By default, both parameters are set to 10^{-2}.

Note: The option `.AugSpace.EnvAlpha` is only necessary when the option `.AugSpace.Method` is set to `'hypercube'`.

2.6 Optimization algorithms

Various configuration options are available regarding the optimization algorithms to solve the RBDO problem. The available optimization methods can be divided into two categories:

- **Gradient-based methods** Currently the interior-point (IP) and sequential quadratic programming (SQP) methods are available. One of these two methods can be set using the field `.Optim.Method`, *e.g.* to use the interior-point method:

```
RBDopt.Optim.Method = 'IP' ;
```

  Additional options specific to these methods can be set by the user. For example, one can set the maximum number of function evaluations for interior-point as follows:

```
RBDopt.Optim.IP.MaxFunEvals = 1000 ;
```

- **Global methods** Currently the genetic algorithm (GA), the constrained $(1+1)$-CMA-ES and their hybrid counterparts are available in UQLAB. To use one of these methods for solving the RBDO optimization problem, *e.g.* the genetic algorithm, one sets:

```
RBDopt.Optim.Method = 'GA' ;
```

  Additional options which are specific to each method exist. For example, when using the genetic algorithm one might need to set a different value of stall generations, say 20. This can be accomplished by setting:

```
RBDopt.Optim.GA.nStall = 20 ;
```

- **General options** Moreover, regardless of the method, some options can be specified manually. For example:

  – The initial search point (not necessary for GA)

```
RBDopt.Optim.StartingPoint = d0;
```

    where `d0` is a row vector whose size is consistent with the design variables defined in the field `.Input.DesVar` (See Section 2.7.1).

  – The number of iterations or generations:

```
RBDopt.Optim.MaxIter = 100;
```

  An exhaustive list of the general optimization options can be found in Table 15.

2.7 Advanced options

2.7.1 Defining the design variables

The design variables can be defined using the field `Input.DesVar`. To entirely define a design variable $\boldsymbol{X}(\boldsymbol{d})$, it is necessary to provide its distribution and standard deviation or coefficient of variation. Remember that the mean value $\boldsymbol{d}$ evolves along the iterations of the RBDO solver. To define for instance a lognormal variable with standard deviation of 1, the following code can be used:

```
RBDopt.Input.DesVar.Type = 'Lognormal' ;
RBDopt.Input.DesVar.Std = 1 ;
```

Alternatively the user can specify a coefficient of variation, *e.g.* for a variable with coefficient of variation of 5%:

```
RBDopt.Input.DesVar.CoV = 0.05 ;
```

In this case, the standard deviation will be computed from the current value of the design in the optimization process.

Note: When the standard deviation or coefficient of variation is set to 0 or when the type is set to `'Constant'`, the corresponding variable is considered deterministic, as in the example of Section 2.1.

Finally, the user can specify the name of the random variable, using the command `.Input.DesVar.Name`. By default, the name of the design variables are `d_1`, `d_2`, etc.

Note: In case of multi-dimensional inputs, each design variables should be defined separately, *e.g.* `RBDopt.Input.DesVar(1)`, `RBDopt.Input.DesVar(2)`, etc.

2.7.2 Defining the cost function

In principle, the cost function defined in the RBDO problem can be any UQLAB MODEL object. In any case the cost is defined in the field `.Cost`. Two options are then possible. One is to define a full model through either a string (`.mString`), a function handle (`.mHanlde`) or an m-file (`.mFile`). For instance, the line of code

```
RBDopt.Cost.mFile = 'uq_mycost';
```

defines the function written in the m-file `uq_myCost.m` as cost. In general, any option available to define a default model as defined in the UQLAB User Manual – the MODEL module can be used in the field `.Cost` (See Table 8 for more details.)

The other option is to use a UQLAB MODEL that is already available in the session. This can be done through the field `.Cost.Model`, *e.g.*

```
RBDopt.Cost.Model = myUQCostModel;
```

where `myUQCostModel` is a MODEL object previously created with the `uq_createModel` command.

Note: The user should make sure that the provided cost model is consistent in its inputs with the design variables defined in the field `.Input.DesVar`.

2.7.3 Parameters mapping

In some cases, it is necessary to define which of the design and environmental parameters affect the computational model and/or in which order. This is achieved through the `.LimitState.PMap` option. The following code:

```
RBDopts.LimitState.PMap.Type = ['d' 'e' 'd' 'e' 'e'];
RBDopts.LimitState.PMap.Index = [1 2 2 3 1];
```

can be used to specify that the computation model takes a 5-dimensional input vector $\boldsymbol{w} = \{d_1, z_2, d_2, z_3, z_1\}$.

Note: For the same input, the default mapping is $\boldsymbol{w} = \{d_1, d_2, z_1, z_2, z_3\}$.

Note: `PMap` can also be used to select subsets of the available design and environmental parameters.

2.7.4 Advanced limit-state function options

While it is normally good practice to define the constraints directly as a UQLAB MODEL object, 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 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 $g(x) \leq T = 5$ defines the failure criterion, one can use the following syntax:

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

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 an RBDO analysis

Syntax

```
myRBDO  = uq_createAnalysis(RBDopts)
```

Input

All the parameters required to define the RBDO analysis are to be given as fields of the structure `RBDopts`. Some of the methods have specific options, that will be reviewed in different tables. The options described in Table 3 are common to all methods.

Table 3: `RBDopts`

●	`.Type`	`'RBDO'`	Identifier of the module. The options corresponding to other types are in the corresponding guides.
☐	`.Method`	String default: `'Two-Level'`	Type of RBDO method. The available options are listed below:
		`'Two-Level'`	Generalized two-level (Section 1.3.1.1).
		`'RIA'`	Reliability index approach (Section 1.3.1.2).
		`'PMA'`	Performance measure approach (Section 1.3.1.3).
		`'QMC'`	Quantile Monte Carlo (Section 1.3.1.4).
		`'SLA'`	Single loop approach (Section 1.3.2).
		`'SORA'`	Sequential optimization and reliability assessment (Section 1.3.3).
☐	`.Name`	String	Name of the module. If not set by the user, a unique string is automatically assigned to it.
●	`.Input`	See Table 4	Specification of the input probabilistic model (design and environmental variables).
●	`.Bounds`	$2 \times M_d$ Double	Bounds of the design space. • The first row corresponds to the lower bounds and the second one to the upper bounds.

⊕	`.TargetPf`	double	Target failure probability ($\bar{P}_f$ in Eq. (1.1)). • Automatically computed if `.TargetBeta` is provided.
⊕	`.TargetBeta`	double	Target reliability index ($\bar{\beta}$ in Eq. (1.3)). • Automatically computed if `.TargetPf` is provided.
●	`.Cost`	See Table 8	Specification of a model for the cost function.
●	`.LimitState`	See Table 9	Specification of the limit-state model
☐	`.Metamodel`	See Table 12	Specification of the metamodel options. If this field is not specified, RBDO will be carried out directly on the computational model defined in `.LimitState`
☐	`.SoftConstraints`	See Table 11	Specification of a model for the soft constraints.
☐	`.Optim`	See Table 15	Specification of the optimization algorithm options.
☐	`.Reliability`	See Table 6	Specification of the reliability method used in the inner loop of the two-level approach.
☐	`.AugSpace`	See Table 13	Specification of the augmented space when using surrogate models in the optimization.
☐	`.Display`	String default: `'standard'`	Level of information displayed by the methods.
		`'quiet'`	Minimum display level, displays nothing or very little 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.

3.1.1 Defining the inputs

The inputs of the optimization problem are defined in the field `.Input`. They consist of the design and environmental variables.

Table 4: `RBDopts.Input`			
●	`.DesVar`	Table 5	Specification of the design variables.
□	`.EnvVar`	INPUT object	Pre-defined INPUT object (See the UQLAB User Manual – the INPUT module for more details on how to build an INPUT object)

The design variables are defined mainly by specifying the marginals. The user is referred to the UQLAB User Manual – the INPUT module for further details on the available distributions. The variability is specified by defining either a standard deviation or a coefficient of variation. Note that with the latter, the variability of the random variable is subject to the actual value of the design variables during the optimization process. The different options are gathered in Table 5.

Table 5: `RBDopts.Input.DesVar`			
●	`.Type`	String	Type of marginal distribution
		`'Constant'`	A constant value
		`'Gaussian'`	Gaussian distribution (See Section A.2 of UQLAB User Manual – the INPUT module)
		`'Lognormal'`	Lognormal distribution (See Section A.3 of UQLAB User Manual – the INPUT module)
		`'Uniform'`	Uniform distribution (See Section A.1 of UQLAB User Manual – the INPUT module)
		`'Exponential'`	Exponential distribution (See Section A.8 of UQLAB User Manual – the INPUT module)
		`'Beta'`	Beta distribution (See Section A.9 of UQLAB User Manual – the INPUT module)
		`'Weibull'`	Weibull distribution (See Section A.6 of UQLAB User Manual – the INPUT module)
		`'Gumbel'`	Gumbel maximum extreme value distribution (See Section A.4 of UQLAB User Manual – the INPUT module)

		`'GumbelMin'`	Gumbel minimum extreme value distribution (See Section A.5 of UQLAB User Manual – the INPUT module)
		`'Gamma'`	Gamma distribution (See Section A.7 of UQLAB User Manual – the INPUT module)
		`'Triangular'`	Triangular distribution (See Section A.10 of of UQLAB User Manual – the INPUT module)
		`'Logistic'`	Logistic distribution (See Section A.11 of of UQLAB User Manual – the INPUT module)
		`'Laplace'`	Laplace distribution (See Section A.12 of UQLAB User Manual – the INPUT module)
		other String	A user-defined marginal type (See Section 2.7 of UQLAB User Manual – the INPUT module)
⊕	`.Std`	Double	Standard deviation of the design variable. • If this value is set to 0, the `.Type` is internally switched to `'constant'`.
⊕	`.CoV`	Double	Coefficient of variation of the design variable. • If this value is set to 0, the `.Type` is internally switched to `'constant'`.
☐	`.Name`	String	Name of the variable

3.1.2 Specifying the reliability method

When using a two-level method, a reliability analysis is performed in the inner loop. All reliability methods available in UQLAB can be used except for AK-MCS. Furthermore, specific options can be set directly within this module as shown in Table 6. The user is referred to the UQLAB User Manual – Structural Reliability for further details on available options and their default values.

Table 6: `RBDopts.Reliability`

☐	`.Method`	String default: `'MCS'`	Reliability analysis method.
		`'MCS'`	Monte Carlo simulation (See Section 1.5.1 in UQLAB User Manual – Structural Reliability)

		`'IS'`	Importance sampling (See Section 1.5.2 in UQLAB User Manual – Structural Reliability)
		`'Subset'`	Subset simulation (See Section 1.5.3 in UQLAB User Manual – Structural Reliability)
		`'FORM'`	First-order reliability method (See Section 1.4.1 in UQLAB User Manual – Structural Reliability)
		`'SORM'`	Second-order reliability method (See Section 1.4.2 in UQLAB User Manual – Structural Reliability)
☐	`.Simulation`	See Table 5 in UQLAB User Manual – Structural Reliability	Options for simulation-based reliability methods. • When using Monte Carlo simulation, `.BatchSize` and `.MaxSampleSize` shall have the same value. In practice only one of the two values should be given. • When using Monte Carlo simulation, the options `.TargetCoV` is ignored by the RBDO module.
☐	`.IS`	See Table 8 in UQLAB User Manual – Structural Reliability	Specific options for importance sampling.
☐	`.Subset`	See Table 9 in UQLAB User Manual – Structural Reliability	Specific options for subset simulation.
☐	`.FORM`	See Table 6 in UQLAB User Manual – Structural Reliability	FORM analysis options used in FORM, SORM and importance sampling.
☐	`.invFORM`	See Table 7	Specific options for inverse FORM.
☐	`.Gradient`	See Table 7 in UQLAB User Manual – Structural Reliability	Finite difference options in gradient-based methods. It applies only for options that use FORM in the reliability methods.

Table 7: RBDOpts.Reliability.invFORM			
☐	.Algorithm	String default: 'AMV'	Algorithm used to find the most probable target point.
		'AMV'	Advanced mean value (See Eq. (1.7)).
		'CMV'	Conjugate mean value (See Eq. (1.8)).
		'HMV'	Hybrid mean value (See Eq. (1.9)).
☐	.StartingPoint	$1 \times M$ Double default: zeros(1,M)	Starting point for the search algorithm.
☐	.StopU	Double default: 10^{-3}	Tolerance value for the optimization algorithm on the design point. The algorithm will stop when $\lvert \boldsymbol{U}_{k+1} - \boldsymbol{U}_k \rvert < StopU$.
☐	.StopG	Double default: 10^{-3}	Tolerance value for the optimization 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 optimization algorithm. If this property should be ignored, it can be set to Inf.

3.1.3 Defining the cost and constraints

The cost function is defined as shown in Table 8. The syntax is the same as for the MODEL except that there is an additional .Model option.

Table 8: RBDopts.Cost			
⊕	.mFile	String	File name of the model function (see Section 2.1.1 in UQLAB User Manual – the MODEL module)
⊕	.mHandle	Function handle	Function handle to be used as model (see Section 2.1.2 in UQLAB User Manual – the MODEL module)
⊕	.mString	String	String containing the model expression (see Section 2.1.3 in UQLAB User Manual – the MODEL module)

⊕	`.Model`	MODEL object of type `'default_model'`, `'metamodel'` or `'uqlink'`	A UQLAB MODEL object available in the session.
☐	`.Parameters`	Any data type	Non-random model parameters (see Section 2.1.1). • Does not apply when using the `.Model` option.
☐	`.isVectorized`	Logical default: true if m-file, false if mString or mHandle	Set to true if the model function supports vectorized input and to false otherwise. • Does not apply when using the `.Model` option.

The limit-state function is defined as a UQLAB MODEL object in the field `.LimitState.Model`. The user can also specify other parameters such as the parameter map and the limit-state options. In fact, to perform a structural reliability analysis, the limit-state function $g(\boldsymbol{x}, \boldsymbol{z})$ is compared to a threshold value, *e.g.* $T = 0$ in the various formulations in Section 1.2. In general though, the thresholds can take any value so that failure is simply defined as $g(\boldsymbol{x}, \boldsymbol{z}) \leq T$. Alternatively, failure can be specified as $g(\boldsymbol{x}, \boldsymbol{z}) \geq T$ by adjustment of the field `RBDopts.LimitState.CompOp` to `'>='`.
The relevant options are summarized in Table 9:

Table 9: `RBDopts.LimitState`

●	`.Model`	UQLAB MODEL	Computational model describing the limit-state function
☐	`.PMap`	See Table 10	Parameter map for the computational model • By default, it is assumed that the computational model takes as input a concatenation of the design and environmental variables in the order given by `.DesVar` and `.EnvVar`.
☐	`.Threshold`	Double default: 0	Threshold T, compared to the limit-state function $g(\boldsymbol{x}, \boldsymbol{z})$.
☐	`.CompOp`	String default: `'<='`	Comparison operator for the limit-state function.
		`'<'`, `'<='`	Failure is defined by $g(\boldsymbol{x}, \boldsymbol{z}) < T$.
		`'>'`, `'>='`	Failure is defined by $g(\boldsymbol{x}, \boldsymbol{z}) > T$.

Table 10: `RBDopts.LimitState.PMap`

☐	`.Type`	$(M_d + M_z)-$dimensional vector	Parameter type of the computational model inputs • The i-th term describes the i-th input parameter: `'d'` stands for design parameter while `'e'` stands for environmental parameter • By default: $\{\underbrace{\texttt{'d'},\ldots,\texttt{'d'}}_{M_d},\underbrace{\texttt{'e'},\ldots,\texttt{'e'}}_{M_z}\}$
☐	`.Index`	Integer	Index of the corresponding design or environmental variable • By default, it is assumed that the computational model takes as input a concatenation of the design and environmental variables in the order given by `.DesVar` and `.EnvVar`.

The soft constraints can be defined similarly to the cost as shown in Table 11.

Table 11: `RBDopts.SoftConstraints`

⊕	`.mFile`	String	File name of the model function (see Section 2.1.1 in UQLAB User Manual – the MODEL module)
⊕	`.mHandle`	Function handle	Function handle to be used as model (see Section 2.1.2 in UQLAB User Manual – the MODEL module)
⊕	`.mString`	String	String containing the model expression (see Section 2.1.3 in UQLAB User Manual – the MODEL module)
⊕	`.Model`	MODEL object of type `'default_model'`, `'metamodel'` or `'uqlink'`	A UQLAB MODEL object available in the session.
☐	`.Parameters`	Any data type	Non-random model parameters (see Section 2.1.1). • Does not apply when using the `.Model` option.
☐	`.isVectorized`	Logical default: true if m-file, false if mString or mHandle	Set to true if the model function supports vectorized input and to false otherwise. • Does not apply when using the `.Model` option.

3.1.4 Surrogate modelling options

A surrogate model is defined by specifying a type using the field `.Metamodel.Type` and the experimental design. The remaining parameters take the default values as indicated in the corresponding modules.

Table 12: `RBDopts.Metamodel`

☐	`.Type`	String default: `'none'`	Specification of the metamodel type. • The metamodel is built automatically by UQLAB in the augmented space • The full model is the computational model built specified in the `.LimitState` field.
		`'Kriging'`	Kriging a.k.a Gaussian process (See UQLAB User Manual – Kriging (Gaussian process modelling))
		`'PCE'`	Polynomial Chaos expansion (See UQLAB User Manual – Polynomial Chaos Expansions)
		`'PC-Kriging'`	Polynomial Chaos - Kriging (See UQLAB User Manual – PC-Kriging)
		`'SVR'`	Support vector regression (See UQLAB User Manual – Support vector machines regression)
		`'LRA'`	Low rank approximation (See UQLAB User Manual – Canonical low-rank approximations)
☐	`.Kriging`	struct	Options of the Kriging metamodel • Only valid when the option `.Metamodel.Type` is set to `'Kriging'`. • All the options of the Kriging metamodel should be set there, especially the metamodel's mandatory options (See UQLAB User Manual – Kriging (Gaussian process modelling) for more information on how to build a Kriging model.)
☐	`.PCE`	struct	Options of the PCE metamodel • Only valid when the option `.Metamodel.Type` is set to `'PCE'`. • All the options of the PCE metamodel should be set there, especially the metamodel's mandatory options (See UQLAB User Manual – Polynomial Chaos Expansions for more information on how to build a PCE model.)

☐	.PCK	struct	Options of the PC-Kriging metamodel • Only valid when the option .Metamodel.Type is set to 'PC-Kriging'. • All the options of the PC-Kriging metamodel should be set there, especially the metamodel's mandatory options (See UQLAB User Manual – PC-Kriging for more information on how to build a PC-Kriging model.)
☐	.SVR	struct	Options of the support vector regression metamodel • Only valid when the option .Metamodel.Type is set to 'SVR'. • All the options of the SVR metamodel should be set there, especially the metamodel's mandatory options (See UQLAB User Manual – Support vector machines regression for more information on how to build a SVR model.)
☐	.LRA	struct	Options of the low rank approximation metamodel • Only valid when the option .Metamodel.Type is set to 'LRA'. • All the options of the LRA metamodel should be set there, especially the metamodel's mandatory options (See UQLAB User Manual – Canonical low-rank approximations for more information on how to build a LRA model.)

The augmented space is built by default internally. The user can however modify some parameters of the augmented space using the .AugSpace.

Table 13: RBDopts.AugSpace

☐	.Method	String default: 'hypercube'	The building scheme of the augmented space.
		'hybrid'	Hybrid: the design variables are sampled in a hypercube while the environmental variables are directly sampled from their given distributions.
		'hypercube'	Hypercube: both the design and environmental variables are sampled in a hypercube.

☐	`.DesAlpha`	Double default: 0.01	Probability of sampling outside the augmented space in each dimension of the design variable space (See Eq. (1.20)). • The smaller the value, the wider the bounds of the augmented space compared to the design space.
☐	`.EnvAlpha`	Double default: 0.01	Probability of sampling outside the augmented space in each dimension of the environmental variable space (See Eq. (1.22)). • Applies only when `.AugSpace.Method` is set to `'hypercube'`.

To enable active learning, at least one of the `.Metamodel.Enrichment` options should be specified.

Table 14: `RBDopts.Constraints.Enrichment`

☐	`.LearningFunction`	String default: `'EFF'` when using Kriging and PC-Kriging `'FBR'` when using PCE `'CMM'` when using SVR and LRA	Learning function for enrichment (See Section 1.4.3)
		`'U'`	Deviation number (See Section 1.4.3.1). • Only valid when using Kriging and PC-Kriging.
		`'EFF'`	Expected feasibility function (See Section 1.4.3.2). • Only valid when using Kriging and PC-Kriging.
		`'CMM'`	Constrained min-max (See Section 1.4.3.3).
		`'FBR'`	Fraction of bootstrap replicates (See Section 1.4.3.4). • Only valid when using SVR.
☐	`.Convergence`	String default: `{'StopDf','StopSign'}`	Convergence criterion of the enrichment procedure. • Multiple criteria can be submitted by listing them in a cell array.
		`'StopLF'`	Stopping criterion on the learning function (Eqs. (1.31), (1.32))
		`'StopDf'`	Stopping criterion on the stability of the failure domain size (Eq. (1.33))
		`'StopSign'`	Stopping criterion on the stability of the sign prediction by successive metamodels (Eq. (1.34))
☐	`.ConvThreshold`	Positive Double default: 0.01	Threshold for the convergence criteria. • When using multiple convergence criteria, specific thresholds can be specified for each by using a cell array.
☐	`.MaxAdded`	Positive Integer default: 100	Maximum number of points to add in the enrichment procedure.
☐	`.Points`	Positive Integer default: 1	Number of points to add per iteration of enrichment.

☐	`.Sampling`	String default: `'LHS'`	Sampling techniques of the candidate set for enrichment ($\mathcal{S}$ in Section 1.4.2). See UQLAB User Manual – the INPUT moduleFor more sampling techniques.
☐	`.SampleSize`	Positive Integer default: 10^4	Size of the candidate set for enrichment ($\mathcal{S}$ in Section 1.4.2).

3.1.5 Optimization algorithms

Table 15: `RBDopts.Optim`			
☐	`.Method`	String default: `'CCMAES'`	Optimization algorithm used in the solution of the RBDO problem.
		`'IP'`	Interior-point using MATLAB's built-in `fmincon` function.
		`'SQP'`	Sequential quadratic programming using MATLAB's built-in `fmincon` function.
		`'CCMAES'`	Constrained $(1+1)$-CMA-ES.
		`'GA'`	Genetic algorithm using MATLAB's built-in `ga` function.
		`'HCCMAES'`	Hybrid constrained $(1+1)$-CMA-ES: CMA-ES followed by a gradient-based refinement of the solution using MATLAB's built-in `fmincon` function.
		`'HGA'`	Hybrid genetic algorithm: GA followed by a gradient-based refinement of the solution using MATLAB's built-in `fmincon` function.
☐	`.StartingPoint`	$1 \times M_d$ or scalar Double default: Center of the search space	Starting point of the optimization process. • If a scalar is given and the problem is multi-dimensional, the given value is replicated over all dimensions.

☐	`.CRN`	Logical default: true	Specify whether common random numbers should be used or not. This option is necessary in a double-loop where a gradient-based algorithm is used in the outer loop associated to a simulation method in the inner loop.
		true	Use common random numbers.
		false	Do not use common random numbers.
☐	`.ConstraintType`	String default: `'Pf'`	Type of constraint that should be used in the formulation of the RBDO problem.
		`'Pf'`	Failure probability: formulation of Eq. (1.1).
		`'Beta'`	Reliability index: formulation of Eq. (1.3).
☐	`.UseLogSpace`	Logical default: true	Specify whether the failure probabilities shall be mapped into the $\log_{10}$-space during the optimization or not.
		true	Map to the $\log_{10}$-space.
		false	Do not map to the $\log_{10}$-space.
⊞	`.IP`	See Table 16	Options relevant to the interior-point optimization method.
⊞	`.SQP`	See Table 17	Options relevant to the sequential quadratic programming optimization method.
⊞	`.CCMAES`	See Table 18	Options relevant to the constrained $(1+1)$-CMA-ES optimization method.
⊞	`.GA`	See Table 19	Options relevant to the genetic algorithm optimization method.

⊞	`.HCCMAES`	See Table 20	Options relevant to the HCCMAES optimization method.
⊞	`.HGA`	See Table 21	Options relevant to the HGA optimization method.
☐	`.Display`	String default: Matches with global display level	Level of information displayed by the methods.
		`'none'`	Minimum display level, displays nothing.
		`'iter'`	Maximum display level, shows detailed information in each iteration of the algorithm.
		`'final'`	Shows information regarding only the convergence of the algorithm, if any.

Table 16: `RBDopts.Optim.IP`

☐	`.StartPoints`	Positive integer default: 1	The number of starting points for the IP algorithm. The default starting point, either given by the user or heuristically calculated, is always considered. The remaining points are sampled uniformly in the search space.
☐	`.MaxFunEvals`	Integer (≥ 1) default: 1000	The maximum number of function evaluations.
☐	`.MaxIter`	Double default: 10^3	Maximum number of iterations allowed in the optimization algorithms.
☐	`.FDType`	String default: `'forward'`	Finite differences scheme used to compute the gradients.
		`'forward'`	Forward finite differences.
		`'central'`	Centered finite differences.
☐	`.FDStepSize`	Double default: 10^{-3}	Scalar or vector step size factor for finite differences. • Use the MATLAB's documentation of `fmincon` to understand how this value is used within MATLAB.
☐	`.FDStep`	String default: `'relative'`	Type of the step size factor.

		'absolute'	Absolute value: the value of .FDStepSize will be used as such.
		'relative'	Relative value: the value of .FDStepSize will be multiplied by the standard deviation of the random design variables. • The option is forced to 'absolute' if the design variables are deterministic.

Table 17: RBDopts.Optim.SQP

☐	.StartPoints	Positive integer default: 1	The number of starting points for the SQP algorithm. The default starting point, either given by the user or heuristically calculated, is always considered. The remaining points are sampled uniformly in the search space.
☐	.MaxFunEvals	Integer (≥ 1) default: 1000	The maximum number of function evaluations.
☐	.MaxIter	Double default: 10^3	Maximum number of iterations allowed in the optimization algorithms.
☐	.FDType	String default: 'forward'	Finite differences scheme used to compute the gradients.
		'forward'	Forward finite differences.
		'central'	Centered finite differences.
☐	.FDStepSize	Double default: 10^{-3}	Scalar or vector step size factor for finite differences. • Use the MATLAB's documentation of fmincon to understand how this value is used within MATLAB.
☐	.FDStep	String default: 'relative'	Type of the step size factor.
		'absolute'	Absolute value: the value of .FDStepSize will be used as such.
		'relative'	Relative value: the value of .FDStepSize will be multiplied by the standard deviation of the random design variables. • The option is forced to 'absolute' if the design variables are deterministic.

Table 18: RBDopts.Optim.CCMAES			
☐	.InitialSigma	$1 \times M_d$ or scalar Double default: $1/3(d_{max} - d_{min})$	Initial value of the global step length of the $(1+1)$-CMA-ES algorithm.
☐	.nStall	Positive integer default: 20	The maximum number of stall generations
☐	.TolFun	Double default: 10^{-3}	Termination tolerance on the cost function: The algorithm stops if the average relative change in the best cost function value over .nStall generations is less than or equal to .TolFun
☐	.TolX	Double default: 10^{-3}	Termination tolerance on X: The algorithm stops if the average relative change in the best solution over .nStall generations is less than or equal to .TolX
☐	.TolSigma	Double default: 10^{-3}	Termination tolerance on step size function: The algorithm stops if the the value of σ becomes smaller than .TolSigma.

Table 19: RBDopts.Optim.GA			
☐	.nPop	Positive integer default: $\min(20, \text{floor}(4 + 3\log M_d))$	The population size of each generation
☐	.nStall	Positive integer default: 50	The maximum number of stall generations

Table 20: RBDopts.Optim.HCCMAES			
☐	.InitialSigma	$1 \times M_d$ or scalar Double default: $1/3(d_{max} - d_{min})$	Initial value of the global step length of the $(1+1)$-CMA-ES algorithm.
☐	.nStall	Positive integer default: 20	The maximum number of stall generations
☐	.TolFun	Double default: 10^{-3}	Termination tolerance on the cost function: The algorithm stops if the average relative change in the best cost function value over .nStall generations is less than or equal to .TolFun
☐	.TolX	Double default: 10^{-3}	Termination tolerance on X: The algorithm stops if the average relative change in the best solution over .nStall generations is less than or equal to .TolX

☐	`.TolSigma`	Double default: 10^{-3}	Termination tolerance on step size function: The algorithm stops if the the value of σ becomes smaller than `.TolSigma`.
☐	`.MaxFunEvals`	Integer (≥ 1) default: 1000	The maximum number of function evaluations in the second part of the algorithm.
☐	`.FDType`	String default: `'forward'`	Finite differences scheme used to compute the gradients in the second part of the algorithm.
		`'forward'`	Forward finite differences.
		`'central'`	Centered finite differences.
☐	`.FDStepSize`	Double default: 10^{-3}	Scalar or vector step size factor for finite differences. • Use the MATLAB's documentation of `fmincon` to understand how this value is used within MATLAB.
☐	`.FDStep`	String default: `'absolute'`	Type of the step size factor.
		`'absolute'`	Absolute value: the value of `.FDStepSize` will be used as such.
		`'relative'`	Relative value: the value of `.FDStepSize` will be multiplied by the standard deviation of the random design variables. • The option is forced to `'absolute'` if the design variables are deterministic.

Table 21: `RBDopts.Optim.HGA`

☐	`.nPop`	Positive integer default: $\min(20, \text{floor}(4 + 3\log M_d))$	The population size of each generation
☐	`.nStall`	Positive integer default: 50	The maximum number of stall generations
☐	`.MaxFunEvals`	Integer (≥ 1) default: 1000	The maximum number of function evaluations in the second part of the algorithm.
☐	`.FDType`	String default: `'forward'`	Finite differences scheme used to compute the gradients in the second part of the algorithm.
		`'forward'`	Forward finite differences.

		`'central'`	Centered finite differences.
☐	`.FDStepSize`	Double default: 10^{-3}	Scalar or vector step size factor for finite differences. • Use the MATLAB's documentation of `fmincon` to understand how this value is used within MATLAB.
☐	`.FDStep`	String default: `'absolute'`	Type of the step size factor.
		`'absolute'`	Absolute value: the value of `.FDStepSize` will be used as such.
		`'relative'`	Relative value: the value of `.FDStepSize` will be multiplied by the standard deviation of the random design variables. • The option is forced to `'absolute'` if the design variables are deterministic.

3.2 Accessing the results

Syntax

```
myRBDO  = uq_createAnalysis(RBDopts)
```

Output

The results are stored in the `myRBDO.Results` structure. The provided information is explained in Table 22.

Table 22: `myRBDO.Results`

`.Xstar`	$1 \times M_d$ Double	Solution of the RBDO problem.
`.Fstar`	Double	Cost corresponding to the solution of the RBDO problem.
`.exitNsg`	Integer	Exit condition of the optimization algorithm.
`.History`	struct	History of the optimization process. See Section 2.3 for more details on the saved information.
`.output`	struct	Details about the optimization. See the corresponding algorithm documentation for more details (either in MATLAB or UQLIB user manual).

.ModelEvaluations	Integer	Number of evaluation of the computational model or its surrogate.

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(myRBDO);
```

Description

`uq_print(myRBDO)` prints a report on the results of the RBDO analysis stored in the object `myRBDO`.

3.3.2 Graphically display the results: uq_display

Syntax

```
uq_display(myRBDO);
```

Description

`uq_display(myRBDO)` creates a visualization of the results of the reliability analysis stored in the object `myRBDO`, if possible. If the problem has multiple hard constraints, information about all the constraints is visualized.

`uq_display(myRBDO, outidx)` creates a visualization of the results of the reliability analysis stored in the object `myRBDO`, if possible. If the problem has multiple hard constraints, only information about the constraints specified in the array `outidx` is visualized.

Examples:

`uq_display(myRBDO, [1 3])` will display the history of the constraints 1 and 3 only.

References

Arnold, D. V. and N. Hansen (2012). A (1+1)-CMA-ES for constrained optimisation. In T. Soule and J. H. Moore (Eds.), *Proc. of the Genetic and Evolutionary Computation Conference 2012 (GECCO 2012)*, pp. 297–304. 13

Au, S. K. and J. L. Beck (2001). Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic Engineering Mechanics 16*(4), 263–277. 21

Basudhar, A. and S. Missoum (2008). Adaptive explicit decision functions for probabilistic design and optimization using support vector machines. *Comput. Struct. 86*(19-20), 1904–1917. 11

Bect, J., D. Ginsbourger, L. Li, V. Picheny, and E. Vazquez (2012). Sequential design of computer experiments for the estimation of a probability of failure. *Stat. Comput. 22*(3), 773–793. 10

Bichon, B. J., M. S. Eldred, L. Swiler, S. Mahadevan, and J. McFarland (2008). Efficient global reliability analysis for nonlinear implicit performance functions. *AIAA Journal 46*(10), 2459–2468. 11, 12

Chateauneuf, A. and Y. Aoues (2008). *Structural design optimization considering uncertainties*, Chapter 9, pp. 217–246. Taylor & Francis. 1, 3, 5

Cho, T. M. and B. C. Lee (2011). Reliability-based design optimization using convex linearization and sequential optimization and reliability assessment method. *Struct. Saf. 33*(1), 42–50. 5

Du, X. and W. Chen (2004). Sequential optimization and reliability assessment method for efficient probabilistic design. *J. Mech. Design 126*(2), 225–233. 7

Dubourg, V., B. Sudret, and J.-M. Bourinet (2011). Reliability-based design optimization using Kriging and subset simulation. *Struct. Multidisc. Optim. 44*(5), 673–690. 2, 8

Echard, B., N. Gayton, and M. Lemaire (2011). AK-MCS: an active learning reliability method combining Kriging and Monte Carlo simulation. *Struct. Saf. 33*(2), 145–154. 10, 12

Hasofer, A.-M. and N.-C. Lind (1974). Exact and invariant second moment code format. *J. Eng. Mech. 100*(1), 111–121. 2

Liang, J., Z. Mourelatos, and J. Tu (2004). A single-loop method for reliability-based design optimization. In *Proc. DETC'04 ASME 2004 Design engineering technical conferences and computers and information in engineering conference, Sept.28 - Oct. 2, 2004, Salt Lake City, Utah, USA*. 6

Marelli, S., R. Schöbi, and B. Sudret (2021). UQLab user manual – Reliability analysis. Technical report, Chair of Risk, Safety & Uncertainty Quantification, ETH Zurich. Report # UQLab-V1.4-107. 2

Marelli, S. and B. Sudret (2018). An active-learning algorithm that combines sparse polynomial chaos expansions and bootstrap for structural reliability analysis. *Struct. Saf. 75*, 67–74. 12

Moustapha, M. (2016). *Adaptive surrogate models for the reliable lightweight design of automotive body structures*. Ph. D. thesis, Université Blaise Pascal, Clermont-Ferrand, France. 16

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

Moustapha, M., B. Sudret, J.-M. Bourinet, and B. Guillaume (2016). Quantile-based optimization under uncertainties using adaptive Kriging surrogate models. *Struct. Multidisc. Optim. 54*(6), 1403–1421. 6, 9

Rackwitz, R. and B. Fiessler (1978). Structural reliability under combined load sequences. *Computers & Structures 9*, 489–494. 4

Spall, J. C. (2003). *Introduction to stochastic search and optimization: Estimation, simulation and control*. John Wiley & Sons. 14

Taflanidis, A. A. and J. L. Beck (2008). Stochastic subset optimization for optimal reliability problems. *Prob. Eng. Mech 23*, 324–338. 8, 14

Valdebenito, A. M. and G. I. Schuëller (2010). A survey on approaches for reliability-based optimization. *Struct. Multidisc. Optim. 42*, 645–663. 1

Wu, Y.-T., Y. Shin, R. Sues, and M. Cesare (2001). Safety-factor based approach for probabilistic-based design optimization. In *42nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Seattle, WA*, pp. 1–9. 7

Youn, B. D., K. K. Choi, and L. Du (2005). Enriched performance measure approach for reliability-based design optimization. *AIAA Journal 43*(4), 874–884. 5