ETH zürich

UQLab User Manual
Stochastic Spectral Embedding

P.-R. Wagner, 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

P.-R. Wagner, S. Marelli, B. Sudret, UQLab user manual – Stochastic Spectral Embedding, Report UQLab-V2.1-118, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland, 2024

BibTeX entry

```
@TechReport{UQdoc_21_118,
author = {Wagner, P.-R. and Marelli, S. and Sudret, B.},
title = {{UQLab user manual -- Stochastic Spectral Embedding}},
institution = {Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich,
Switzerland},
year = {2024},
note = {Report UQLab-V2.1-118}
}
```

Document Data Sheet

Document Ref.	UQLAB-V2.1-118
Title:	UQLAB user manual – Stochastic Spectral Embedding
Authors:	P.-R. Wagner, S. Marelli, B. Sudret Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland
Date:	15/04/2024

Doc. Version	Date	Comments
V2.1	15/04/2024	UQLab release 2.1
V2.0	01/02/2022	First release

Abstract

Stochastic spectral embedding (SSE) is an advanced surrogate modelling technique for models with inhomogeneous complexity, i.e. models that show different degrees of complexity depending on the region of the input parameter space they are evaluated at. The technique draws its strength from constructing a sequence of residual spectral expansions of the target model in subdomains of the input space.

The UQLAB stochastic spectral embedding module offers an easy way to use SSE surrogate models based on static and sequential experimental designs. It makes use of other UQLAB modules (UQLAB User Manual – the INPUT module, UQLAB User Manual – the MODEL module) to define the input distribution and the original model. It also relies on the UQLAB PCE-module (UQLAB User Manual – Polynomial Chaos Expansions) for constructing the residual expansions. This manual is divided into three parts:

- An introduction to the main ideas and theoretical foundations of SSE;
- An example-based guide to SSE with an explanation of available options and methods;
- A comprehensive reference list detailing all available functionalities of the SSE module.

Keywords: UQLAB, Stochastic spectral embedding, metamodelling, spectral expansions

Contents

1 Theory **1**

1.1 Introduction 1

1.2 Stochastic spectral embedding 1

1.2.1 Quantile and physical space 4

1.2.2 SSE construction in practice 6

1.3 Sequential partitioning algorithm 6

1.3.1 Refinement domain selection 6

1.3.2 Partitioning strategy 8

1.3.3 Sequential experimental design 8

1.3.4 Residual expansion 9

1.3.5 Post-expansion operation 10

1.3.6 Stopping criteria 10

1.4 SSE data structure 11

1.5 *A-posteriori* error estimation 11

1.5.1 Normalized empirical error 12

1.5.2 Weighted residual expansion error 12

1.6 Post-processing an SSE 13

1.6.1 Moments of an SSE 13

1.6.2 Bayesian inversion 14

1.6.3 Reliability analysis 15

2 Usage **17**

2.1 Reference problem . . . 17

2.2 Problem set-up . . . 17

2.2.1 Initialize UQLAB . . . 17

2.2.2 Defining the full model . . . 18

2.2.3 Defining the input distribution . . . 18

2.3 Constructing the SSE metamodel . . . 18

2.3.1 Initializing the metamodel . . . 18

2.3.2 Analytical output moment computation . . . 19

2.3.3 Experimental design . . . 19

2.3.4 Creating the metamodel . . . 19

2.4 Assessing the metamodel . . . 20

2.4.1 Use of validation set . . . 20

2.4.2 Additional display functionality . . . 22

2.5 Evaluating the metamodel . . . 24

2.6 Vector-valued models . . . 24

2.7 Constant parameters . . . 25

2.8 Flattened representation . . . 25

2.9 Advanced options for SSE metamodel . . . 25

2.9.1 Refinement domain selection . . . 26

2.9.2 Partitioning strategy . . . 26

2.9.3 Sequential experimental design . . . 27

2.9.4 Residual expansion . . . 28

2.9.5 Post-expansion operation . . . 28

2.9.6 Stopping criterion . . . 28

3 Reference List **31**

3.1 Create the SSE metamodel . . . 33

3.1.1 Experimental design options 34
3.1.2 Validation Set 35
3.1.3 Refinement options 35
3.1.4 Post-processing options 35
3.1.5 Stopping criterion 35
3.2 Custom function handle specifications 36
3.2.1 uq_SSE_postExpansion 36
3.2.2 uq_SSE_partitioning 36
3.2.3 uq_SSE_enrichment 37
3.2.4 uq_SSE_refineScore 37
3.2.5 uq_SSE_stopping 37
3.3 Accessing the results 38
3.4 Evaluating the metamodel (predictor) 40

Chapter 1

Theory

1.1 Introduction

Surrogate modelling (or metamodelling) techniques provide means to replace a computationally expensive numerical model with a cheaper-to-evaluate replacement in many UQ analyses. Most surrogate modelling techniques (e.g. polynomial chaos expansion UQLAB User Manual – Polynomial Chaos Expansions or kriging, UQLAB User Manual – Kriging (Gaussian process modelling)) tend to perform well on models that show homogeneous complexity throughout the input space. Some models of practical engineering relevance, however, can show a highly localised behaviour in different regions of the input space. Relevant examples include likelihood functions used in Bayesian inference (Nagel and Sudret, 2016), crash test simulations (Serna and Bucher, 2009), snap-through models (Hrinda, 2010), and discontinuous models in general.

Stochastic spectral embedding (SSE, Marelli et al. (2021)) is a surrogate modelling technique that combines spectral representations and adaptive domain decompositions. The former ensure a degree of globality, while the latter give SSE its local approximation strength.

The SSE module of UQLAB provides a flexible implementation of the sequential partitioning algorithm (Marelli et al., 2021), that allows easy customization of the code for further research and applications to various UQ problems. The implementation also makes use of the powerful UQLAB PCE module (UQLAB User Manual – Polynomial Chaos Expansions) for the local residual expansions.

1.2 Stochastic spectral embedding

We will consider herein random variables of the form $Y = \mathcal{M}(\boldsymbol{X})$ that have finite second moments (i.e. $Y^2 < \infty$), where $\boldsymbol{X}$ is an $M-$dimensional random vector with joint distribution

$\boldsymbol{X} \sim f_{\boldsymbol{X}}$. These random variables admit a spectral decomposition of the form:

$$\mathcal{M}(\boldsymbol{X}) = \sum_{i \in \mathbb{N}} a_i \Psi_i(\boldsymbol{X}). \tag{1.1}$$

For practical computations, this infinite series is typically truncated to:

$$\mathcal{M}(\boldsymbol{X}) \approx \mathcal{M}_{\mathrm{S}}(\boldsymbol{X}) = \sum_{j \in \mathcal{A}} a_j \Psi_j(\boldsymbol{X}), \tag{1.2}$$

where $\mathcal{A}$ is a truncation set (typically related to the complexity of the basis functions, e.g. maximum frequency in Fourier expansions, or maximum polynomal degree in PCE). Because of this, the expansion is in general not exact, hence we define the *residual* $\mathcal{R}(\boldsymbol{X})$ as:

$$\mathcal{R}(\boldsymbol{X}) = \mathcal{M}(\boldsymbol{X}) - \mathcal{M}_{\mathrm{S}}(\boldsymbol{X}). \tag{1.3}$$

Spectral expansions belong to the class of global representations, i.e. the basis functions in Eq. (1.2) have support on the entire domain $\mathcal{D}_{\boldsymbol{X}}$. Therefore, highly localised models, or those with inhomogeneous behaviour throughout the input domain, tend to require an extremely large number of terms in the truncated expansion to achieve satisfactory approximation accuracy (Gibbs phenomenon). As an example, the number of terms in the well-established PCE (UQLAB User Manual – Polynomial Chaos Expansions) can grow very fast when the underlying model has strongly localised behaviour, because a high polynomial degree is required for an accurate representation.

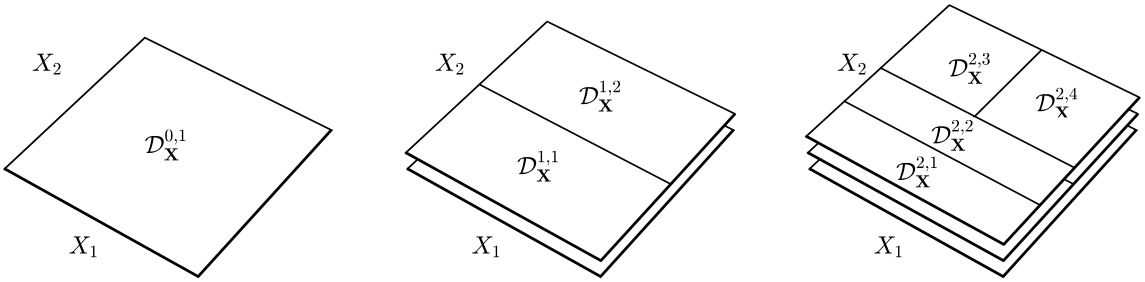
To alleviate this limitation, while still capitalising on the powerful convergence properties of spectral methods, SSE constructs a sequence of spectral expansions of manageable complexity on increasingly smaller subdomains of the input domain. Such subdomains are denoted $\mathcal{D}_{\boldsymbol{X}}^{\ell,p} \subseteq \mathcal{D}_{\boldsymbol{X}}$, where ℓ is the expansion level and p is a subdomain index within the level. The expansion is performed only on the local residual from the previous level.

For illustration purposes, Figure 1 shows an example of sequential partitioning for a simple two-dimensional bounded domain, obtained by splitting each subdomain into two equal subdomains along a random dimension. When $\ell = 0$, there is only a single subdomain $\mathcal{D}_{\boldsymbol{X}}^{0,1} \stackrel{\text{def}}{=} \mathcal{D}_{\boldsymbol{X}}$ and the residual is $\widehat{\mathcal{R}}_{\mathrm{S}}^{0,1}(\boldsymbol{X}) = \mathcal{M}_{\mathrm{S}}(\boldsymbol{X})$ from Eq. (1.3).

In more formal terms, SSE in its basic form can be written as a multi-level expansion of the form:

$$\mathcal{M}_{\mathrm{SSE}}(\boldsymbol{X}) = \sum_{\ell=0}^{L} \sum_{p=1}^{P_\ell} \mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^{\ell,p}}(\boldsymbol{X})\, \widehat{\mathcal{R}}_{\mathrm{S}}^{\ell,p}(\boldsymbol{X}), \tag{1.4}$$

where L is the total number of expansion levels considered, P_ℓ is the number of subdomains at the level $\ell \in \{1, \ldots, L\}$, $\mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^{\ell,p}}(\boldsymbol{X})$ is the indicator function of the subdomain $\mathcal{D}_{\boldsymbol{X}}^{\ell,p}$. Finally,



(a) $\ell = 0$ (b) $\ell = 1$ (c) $\ell = 2$

Figure 1: Example of a possible partitioning sequence for an SSE with $L = 2$ and $P_\ell = 2^\ell$, and bounded uniform marginal distributions. In this simple example, each subdomain is split into two equal parts along a random direction.

$\widehat{\mathcal{R}}_{\mathrm{S}}^{\ell,p}(\boldsymbol{X})$ is the truncated expansion of the residual of the SSE up to level $\ell - 1$:

$$\mathcal{R}^{\ell}(\boldsymbol{X}) = \mathcal{M}(\boldsymbol{X}) - \sum_{m=0}^{\ell-1} \sum_{p=1}^{P_m} \mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^{m,p}}(\boldsymbol{X}) \widehat{\mathcal{R}}_{\mathrm{S}}^{m,p}(\boldsymbol{X}). \tag{1.5}$$

In the above equations, each residual term is expanded onto a local orthonormal basis as follows:

$$\widehat{\mathcal{R}}_{\mathrm{S}}^{m,p}(\boldsymbol{X}) = \sum_{j \in \mathcal{A}^{m,p}} a_j^{m,p} \Psi_j^{m,p}(\boldsymbol{X}). \tag{1.6}$$

A local inner product is defined in the domain $\mathcal{D}_{\boldsymbol{X}}^{m,p}$:

$$\left\langle \Psi_i^{m,p}(\boldsymbol{X}), \Psi_j^{m,p}(\boldsymbol{X}) \right\rangle_{m,p} \overset{\text{def}}{=} \int_{\mathcal{D}_{\boldsymbol{X}}^{m,p}} \Psi_i^{m,p}(\boldsymbol{x}) \Psi_j^{m,p}(\boldsymbol{x}) f_{\boldsymbol{X}}^{m,p}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}, \tag{1.7}$$

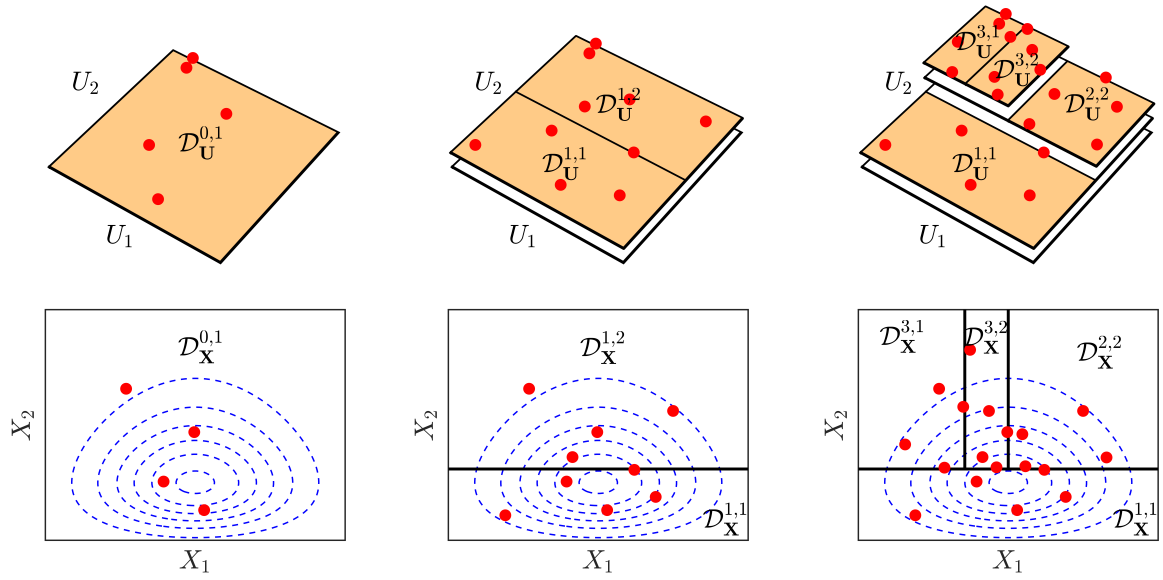
where

$$f_{\boldsymbol{X}}^{m,p}(\boldsymbol{x}) = \mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^{m,p}}(\boldsymbol{x}) \frac{f_{\boldsymbol{X}}(\boldsymbol{x})}{\mathcal{V}^{m,p}} \tag{1.8}$$

is the joint PDF of the input parameters restricted to the subdomain $\mathcal{D}_{\boldsymbol{X}}^{m,p}$ and rescaled by its probability mass $\mathcal{V}^{m,p}$:

$$\mathcal{V}^{m,p} = \int_{\mathcal{D}_{\boldsymbol{X}}^{m,p}} f_{\boldsymbol{X}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}. \tag{1.9}$$

A crucial aspect of SSE is that by partitioning the entire input domain $\mathcal{D}_{\boldsymbol{X}}$ into smaller subdomains $\mathcal{D}_{\boldsymbol{X}}^{\ell,p}$, it trades the complexity of the single, often global expansion in Eq. (1.2) for a (possibly large) number of local expansions with much smaller truncation sets. In cases where the spectral basis is continuous in Eq. (1.2), SSE results in a final piecewise continuous approximation, but no continuity is ensured across the boundaries of the subdomains. Mean-square convergence of the procedure is guaranteed by the spectral convergence in each level, which implies that the residual local variance in each subdomain is in expectation de-



(a) Initialisation (b) First iteration (c) 3rd iteration

Figure 2: Graphical representation of a data-driven SSE construction for a two-dimensional problem with independent prior distributions. Upper row: partitioning in the quantile space; Lower row: partitioning in the unbounded physical space with $f_{\boldsymbol{X}}$ contour lines in dashed blue. Red dots show a sequential experimental design that is enriched at every refinement step with $N_{\text{enr}} = 3$ new sample points in each created subdomain. The terminal domains $\mathcal{T}$ are highlighted in orange. The splitting direction in each subdomain is determined randomly in this example.

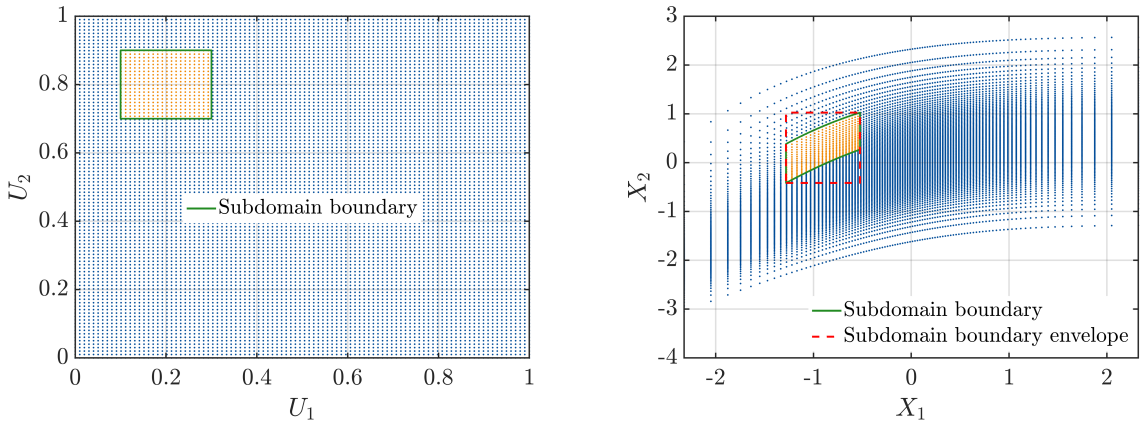
creasing rapidly. In other words, for each increasing level ℓ in Eq. (1.4), new discontinuity bounds are generated during the partitioning step, but the variance of the overall residual is reduced, thus resulting, in expectation, in lower amplitude discontinuities. This behaviour is analogous to that of regression trees (Friedman, 1991; Breiman, 2017).

1.2.1 Quantile and physical space

To extend SSE to the case of unbounded distributions $f_{\boldsymbol{X}}$, and for general ease of bookkeeping, we manage the entire SSE construction in the *quantile space*. Under very general conditions, it is possible to bijectively map any random vector $\boldsymbol{X}$ with joint distribution $F_{\boldsymbol{X}}$ living in the original or *physical space* $\mathcal{D}_{\boldsymbol{X}}$, to a random vector with uniform independent components $\boldsymbol{U} \sim \mathcal{U}(0,1)^M$ living in the quantile space $\mathcal{D}_{\boldsymbol{U}}$ through an appropriate probability transform (Rosenblatt, 1952; Torre et al., 2019b):

$$\begin{aligned} \boldsymbol{U} &= \mathfrak{T}(\boldsymbol{X}) \\ \boldsymbol{X} &= \mathfrak{T}^{-1}(\boldsymbol{U}), \end{aligned} \tag{1.10}$$

where $\mathfrak{T}$ and $\mathfrak{T}^{-1}$ denote the probability transform and its inverse, respectively. This mapping is shown graphically in Figure 2.



(a) Quantile space (b) Physical space

Figure 3: *Stochastic spectral embedding*: Illustration of how dependence is handled in the algorithm. For two mutually dependent input parameters X_1 and X_2, the plots show a set of sample points in the physical and quantile space with a subset of points inside an exemplary subdomain. The parameter dependence leads to a loss of orthogonality of the subdomains in the physical space, which necessitates the introduction of an *enveloping hypercube* in the physical space to construct the spectral basis functions $\Psi^k_{\boldsymbol{\alpha}}$.

Generally, the input random vector $\boldsymbol{X}$ may have mutually dependent marginals that we treat with copula theory in UQLAB (UQLAB User Manual – the INPUT module). Dependence is challenging for the construction of the spectral basis functions $\Psi^k_{\boldsymbol{\alpha}}$ in Eq. (1.6). However, as extensively studied in Torre et al. (2019a) for polynomial chaos expansions, regression-based spectral decompositions seem to be most accurate for predictive purposes when dependence is simply ignored and the basis is derived by tensor product of univariate bases orthogonal to the marginals. Applying this to SSE suggests, it is best to construct the domain-wise polynomial basis by assuming the domain-wise marginals to be independent.

As the partitioning into orthogonal subdomains is defined in the independent quantile space (see Figure 2), the transformation of those domains to the dependent physical space results in non-orthogonal domains (see Figure 3). Constructing the univariate polynomial bases requires knowledge of the marginal bounds in the physical space, which are not straightforward to compute.

To approximate these bounds, we therefore (1) randomly sample the $(M-1)$-dimensional boundary of the hypercube in the quantile space, (2) transform those points to the physical space using a suitable probability transform (see Eq. (1.10)) and proceed to (3) compute the basis with an orthogonal *enveloping hypercube* in the physical space around those transformed points. This process is depicted graphically in Figure 3b.

1.2.2 SSE construction in practice

For it to be useful in practice, SSE needs to be trainable from a finite-size experimental design. Hereinafter, we consider the experimental design $\mathcal{X} = \left\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(N_\mathcal{X})}\right\}$ and corresponding model evaluations in $\mathcal{Y} = \left\{\mathcal{M}(\boldsymbol{x}^{(1)}), \cdots, \mathcal{M}(\boldsymbol{x}^{(N_\mathcal{X})})\right\}$ as the only data available for training.

The SSE equation in Eq. (1.4) is most suitable for *full* SSEs, i.e. SSEs where each level ℓ contains residual expansions covering the full input domain. However, in a data-driven setting it is often more useful to construct expansions only where data are available, thus giving rise to *sparse stochastic spectral embeddings* (see Figure 2). Instead of Eq. (1.4), these are more compactly written utilizing a set of multi-indices $\mathcal{K} \subseteq \mathbb{N}^2$, by

$$\mathcal{M}_{\text{SSE}}(\boldsymbol{X}) = \sum_{k \in \mathcal{K}} \mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^k}(\boldsymbol{X})\, \widehat{\mathcal{R}}_{\text{S}}^k(\boldsymbol{X}). \tag{1.11}$$

In this equation $k = (\ell, p)$ denotes the levels $\ell \in \{0, \ldots, L\}$ and level-specific index $p \in \{1, \ldots, P_\ell\}$, where L is the number of levels and P_ℓ is the number of subdomains at the ℓ-th level.

This compact notation easily distinguishes between two types of domains: (1) standard already partitioned domains and (2) *terminal* domains that have not been partitioned yet. This distinction is useful for selecting a refinement domain and defining suitable error measures. Formally, by gathering the multi-indices of the terminal domains in the set $\mathcal{T}$, Eq. (1.11) can be rewritten to

$$\mathcal{M}_{\text{SSE}}(\boldsymbol{X}) = \sum_{k \in \mathcal{K} \setminus \mathcal{T}} \mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^k}(\boldsymbol{X})\, \widehat{\mathcal{R}}_{\text{S}}^k(\boldsymbol{X}) + \sum_{k \in \mathcal{T}} \mathbf{1}_{\mathcal{D}_{\boldsymbol{X}}^k}(\boldsymbol{X})\, \widehat{\mathcal{R}}_{\text{S}}^k(\boldsymbol{X}), \tag{1.12}$$

1.3 Sequential partitioning algorithm

Algorithmically, SSE consists of a local refinement sequence of a global spectral expansion into sequentially smaller subdomains $\mathcal{D}_{\boldsymbol{X}}^k$. This *sequential partitioning algorithm* is sketched in an abbreviated form in Algorithm 1. We will introduce the individual steps of the algorithm in the following sections.

1.3.1 Refinement domain selection

At every step, the sequential partitioning algorithm chooses a *refinement domain* from the set of terminal domains $\mathcal{T}$ to be split and possibly enriched with sample points. To choose a domain, a *refinement score* $\mathcal{E}^k$ is devised for each terminal domain and the domain with the

Algorithm 1 Sequential partitioning algorithm

Initialisation

1. Sample an initial experimental design from the input distribution
2. Calculate initial expansion

Select refinement domain: see Section 1.3.1

1. Partition domain see Section 1.3.2
2. **For each subdomain**:
   (a) Enrich sample see Section 1.3.3
   (b) Create residual expansion see Section 1.3.4
   (c) Perform post-expansion operations see Section 1.3.5
   (d) Update refinement score see Section 1.3.1
3. If stopping criterion is not met, iterate see Section 1.3.6

largest refinement score is refined in the next step

$$k_{\text{refine}} = \underset{k \in \mathcal{T}}{\arg\max}\, \mathcal{E}^k. \tag{1.13}$$

A straight-forward refinement score is obtained based on the k-th domain importance to the overall approximation based on

$$\mathcal{E}^{\ell+1,s} = \begin{cases} E_{\text{LOO}}^{\ell+1,s} \mathcal{V}^{\ell+1,s}, & \text{if} \quad \exists\, \widehat{\mathcal{R}}_{\text{S}}^{\ell+1,s}, \\ E_{\text{LOO}}^{\ell,s} \mathcal{V}^{\ell+1,s}, & \text{otherwise.} \end{cases} \tag{1.14}$$

This estimator incorporates the subdomain size through the input probability mass $\mathcal{V}^{\ell+1,s}$, and the approximation accuracy, through the leave-one-out cross-validation error estimator of the residual expansion. The distinction is necessary to assign an error estimator also to domains that have too few points to construct a residual expansion, in which case the error estimator of the previous level $E_{\text{LOO}}^{\ell,s}$ is reused (see Eq. (1.27)).

In practice, it can be useful to restrict the refinement candidate domains to a subset of the terminal domains. Eq. (1.13) is then modified to

$$k_{\text{refine}} = \underset{k \in \tilde{\mathcal{T}}}{\arg\max}\, \mathcal{E}^k, \tag{1.15}$$

where the *refinement candidate set* $\tilde{\mathcal{T}} \subseteq \mathcal{T}$ is used to exclude certain domains from further refinement. A possible choice for this purpose is given next.

1.3.1.1 Relative refinement score threshold

A possible refinement candidate set can be constructed by introducing a relative refinement score threshold τ_{ref}. With it, only terminal domains with a relative refinement score above

τ_{ref} are included in $\tilde{\mathcal{T}}$. The relative refinement score ε^k is computed for each terminal domain with

$$\varepsilon^k \overset{\text{def}}{=} \frac{\mathcal{E}^k}{\sum_{k\in\mathcal{T}} \mathcal{E}^k}, \tag{1.16}$$

and the refinement candidate set is then given by

$$\tilde{\mathcal{T}} = \left\{ k \in \mathcal{T} : \varepsilon^k \geq \tau_{\text{ref}} \right\}. \tag{1.17}$$

Introducing a relative refinement score threshold τ_{ref} reduces the set of refinement candidates. This way, it can be used to terminate the algorithm if $\tilde{\mathcal{T}} = \emptyset$, as discussed in Section 1.3.6.2.

1.3.2 Partitioning strategy

After selecting a refinement domain, it is necessary to devise a way to partition that domain. A straight-forward way to partitioning is by splitting the refinement domain into two parts of equal probability mass along one of the input directions, $d^k \in \{1, \cdots, M\}$. Note that the direction in itself can be different for each subdomain, even on the same level.

In the general case, different strategies can be employed to choose a specific splitting direction d^k. Different heuristic reasoning can be used to make this choice, including purely random splitting (as in Figure 2), using the direction of maximum residual difference, or estimates of the variability of the $\widehat{\mathcal{R}}_{\text{S}}^k$ in each direction (following the same rationale as in Shields (2018)). The optimal criterion can be application-specific and may in principle depend on the chosen spectral representation.

1.3.2.1 Sobol' index-based partitioning

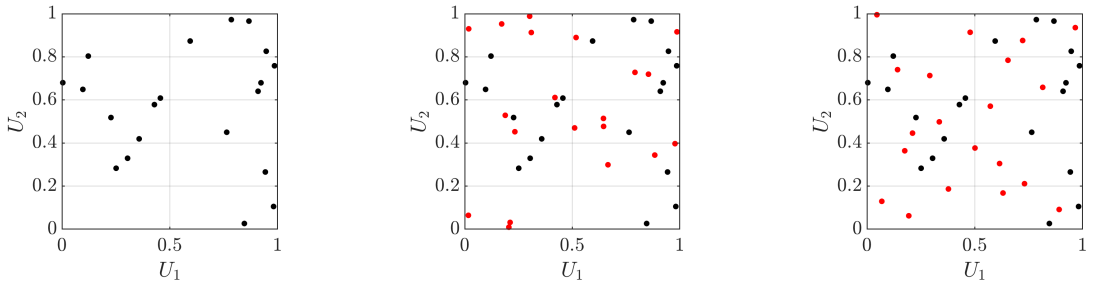
One way to partition domains is by splitting according to the direction of highest variability of $\widehat{\mathcal{R}}_{\text{S}}^k$. We therefore split the refinement subdomain $\mathcal{D}_{\boldsymbol{X}}^{k_{\text{refine}}}$ into two subdomains,i.e. $N_S = 2$, with equal probability mass along the direction that has the maximum first order Sobol' index (Sobol', 1993)

$$d^k = \underset{i\in\{1,\cdots,M\}}{\arg\max}\ S_i^k. \tag{1.18}$$

For PCE residual expansions, the first-order Sobol' indices S_i^k can be analytically derived from the expansion coefficients of $\widehat{\mathcal{R}}_{\text{S}}^k$ (Sudret, 2008).

1.3.3 Sequential experimental design

To increase the number of informative model evaluations, it is possible to sequentially enrich the experimental design. Particularly, after partitioning the refinement domain, N_{enr}



(a) Initial sample (b) Uninformed enrichment (c) Informed enrichment

Figure 4: Example of two uniform enrichment schemes in the quantile space: The black points are the initial sample and the red points the newly added sample points. The second plot displays an uninformed enrichment that does not consider the initial sample. The third plot displays an enrichment strategy that considers the initial sample and provides more space-filling and thus informative new sample points.

additional points are added to the experimental design through a suitable strategy.

A simple strategy is to uniformly enrich the experimental design in the newly created subdomains according to the conditional input distribution. To increase the information content of newly added sample points, it is beneficial to choose an enrichment scheme that considers the already available experimental design and increases the space-filling properties of the enriched sample as shown in Figure 4.

At the first iteration, to construct the initial expansion, the enrichment occurs randomly in the entire input quantile space. We observed from repeated tests that the algorithm is less stable if the initial expansion is less accurate. To account for this, we start out with an initial experimental design of size $2N_{\text{enr}}$ and switch to an enrichment of N_{enr} in all subsequent iterations.

1.3.4 Residual expansion

Different spectral techniques can be used to construct the residual expansions in Eq. (1.6) (e.g. Fourier series, polynomial chaos expansion). The most widely used spectral expansion technique today is polynomial chaos expansion (PCE, UQLAB User Manual – Polynomial Chaos Expansions) that is currently the only supported residual expansion technique in UQLAB, as it also allows to analytically derive numerous quantities of interest as detailed in Section 1.6.

If PCE is used, the residual expansions in Eq. (1.6) read

$$\widehat{\mathcal{R}}^k_{\text{PCE}}(\boldsymbol{X}) = \sum_{\boldsymbol{\alpha}\in\mathcal{A}^k} a^k_{\boldsymbol{\alpha}} \Psi^k_{\boldsymbol{\alpha}}(\boldsymbol{X}), \tag{1.19}$$

where $\boldsymbol{\alpha}$ is a multi-index denoting the dimension-wise polynomial degree of the basis func-

tions $\Psi_{\boldsymbol{\alpha}}^k$ and $\mathcal{A}^k$ is a truncation set as detailed in UQLAB User Manual – Polynomial Chaos Expansions.

1.3.5 Post-expansion operation

After the expansion, it can be useful to perform additional operations with the newly computed expansion. This includes the computation of additional metrics (e.g. conditional failure probability in a subdomain) or correcting the expansion to account for large discontinuities at domain boundaries.

1.3.6 Stopping criteria

There are different ways the sequential partitioning algorithm may be terminated. This section provides an overview of all criteria the algorithm checks. If either of the following criteria are met, the algorithm is stopped.

1.3.6.1 Maximum number of refinement steps

The algorithm is terminated after a fixed number of refinement steps. This criterion can be used to cap the maximum cost of the algorithm.

1.3.6.2 No more refinement domains found

As discussed in Section 1.3.1, it is possible to construct a reduced refinement candidate set $\tilde{\mathcal{T}}$ using different strategies (e.g. τ_{ref}). The algorithm automatically stops when these strategies yield an empty refinement set $\tilde{\mathcal{T}} = \emptyset$, and refinement is thus no longer possible.

1.3.6.3 No more expansions possible

One way to ensure a minimum quality of the constructed residual expansions is by allowing expansions only in domains that contain at least a fixed number of sample points. This required *number of expansion sample points* is denoted by N_{exp} and a residual expansion $\widehat{\mathcal{R}}_{\text{S}}^k$ is then constructed only if

$$N_{\mathcal{X}}^k \geq N_{\text{exp}}, \tag{1.20}$$

where $N_{\mathcal{X}}^k$ is the number of sample points inside the k-th domain.

If no more expansions can be constructed in any terminal domain $k \in \mathcal{T}$, the algorithm is terminated.

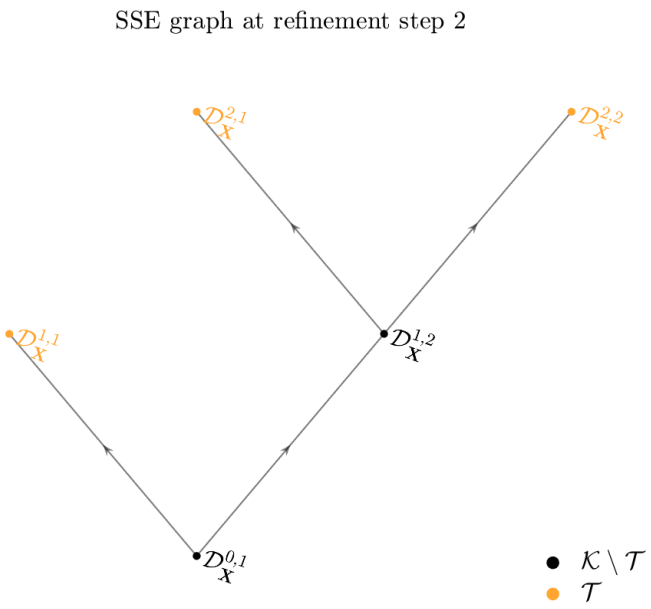


Figure 5: A visualization of the directed tree SSE graph used to store the SSE structure.

1.4 SSE data structure

SSE objects in UQLAB use graphs to model the relationships between subdomains at different levels. An SSE *graph* is a structure consisting of nodes and edges, where each node corresponds to one subdomain, and the edges encode the relationship between subdomains. Particularly, because the order of levels imposes a hierarchy, and the fact that no subdomain can be its own successor, the SSE graph is a *directed acyclic graph* or more specifically a *directed tree* as shown in Figure 5.

Additionally to representing subdomains, the nodes are used to store all subdomain-related information such as their bounds, residual expansions, and the domain refinement scores. The edges, on the other hand, are only used for encoding the relationship between subdomains.

1.5 *A-posteriori* error estimation

After constructing the SSE surrogate model, assessing it accuracy is an important task. Arguably the best known accuracy estimator in function approximation is the so-called generalisation error E_{gen}, which for SSE is given by

$$E_{\text{gen}} \overset{\text{def}}{=} \mathbb{E}\left[\left(\mathcal{M}(\boldsymbol{X}) - \mathcal{M}_{\text{SSE}}(\boldsymbol{X})\right)^2\right]. \tag{1.21}$$

In practice, one often prefers the non-dimensional relative generalization error ϵ_{gen}, that is defined as:

$$\epsilon_{\text{gen}} = \frac{E_{\text{gen}}}{\text{Var}\left[\mathcal{M}(\boldsymbol{X})\right]} \tag{1.22}$$

If, additionally to the training set used to construct the metamodel, an independent set of inputs $\mathcal{X}_{\text{val}} = \left\{\boldsymbol{x}_{\text{val}}^{(1)}, \cdots, \boldsymbol{x}_{\text{val}}^{(N_{\mathcal{X}_{\text{val}}})}\right\}$ and corresponding outputs (also called *validation set*) $\{\mathcal{X}_{\text{val}},\ \mathcal{M}(\mathcal{X}_{\text{val}})\}$ is available, the *validation error* can be calculated as:

$$\epsilon_{\text{val}} = \frac{N_{\mathcal{X}_{\text{val}}} - 1}{N_{\mathcal{X}_{\text{val}}}} \left[\frac{\sum\limits_{\boldsymbol{x}\in\mathcal{X}_{\text{val}}} \left(\mathcal{M}(\boldsymbol{x}) - \mathcal{M}_{\text{SSE}}(\boldsymbol{x})\right)^2}{\sum\limits_{\boldsymbol{x}\in\mathcal{X}_{\text{val}}} \left(\mathcal{M}(\boldsymbol{x}) - \hat{\mu}_{Y_{\text{val}}}\right)^2}\right] \tag{1.23}$$

where $\hat{\mu}_{Y_{\text{val}}} = \frac{1}{N_{\mathcal{X}_{\text{val}}}} \sum_{\boldsymbol{x}\in\mathcal{X}_{\text{val}}} \mathcal{M}(\boldsymbol{x})$ is the sample mean of the validation set response. This version of the relative generalization error is useful to compare the performance of different surrogate models when evaluated on the same validation set.

If no validation set is available, as commonly the case with expensive computational models, there are two main ways to estimate ϵ_{gen} with SSE: *normalized empirical error* and *weighted residual expansion error*.

1.5.1 Normalized empirical error

The normalized empirical error ϵ_{emp} is an estimator of the generalization error based on the accuracy with which the metamodel reproduces the experimental design model evaluations $\mathcal{M}(\mathcal{X})$. It is given by:

$$\epsilon_{\text{emp}} = \frac{\sum\limits_{\boldsymbol{x}\in\mathcal{X}} \left(\mathcal{M}(\boldsymbol{x}) - \mathcal{M}_{\text{SSE}}(\boldsymbol{x})\right)^2}{\sum\limits_{\boldsymbol{x}\in\mathcal{X}} \left(\mathcal{M}(\boldsymbol{x}) - \hat{\mu}_Y\right)^2} \tag{1.24}$$

where $\hat{\mu}_Y = \frac{1}{N_{\mathcal{X}}} \sum_{\boldsymbol{x}\in\mathcal{X}} \mathcal{M}(\boldsymbol{x})$ is the sample mean of the experimental design response.

1.5.2 Weighted residual expansion error

If we denote the residual expansion generalization error in the k-th domain by

$$E_{\text{gen}}^k = \mathbb{E}\left[\left(\mathcal{R}^k(\boldsymbol{X}^k) - \widehat{\mathcal{R}}_{\text{S}}^k(\boldsymbol{X}^k)\right)^2\right], \tag{1.25}$$

then the global generalisation error is simply given by the average error in each terminal domain, which is equal to the sum of the terminal domain errors weighted by the corresponding probability mass in Eq. (1.9):

$$E_{\text{gen}} = \mathbb{E}\left[E_{\text{gen}}^k\right] = \sum_{k\in\mathcal{T}} E_{\text{gen}}^k \cdot \mathcal{V}^k. \tag{1.26}$$

An estimator for regression-based spectral expansion methods is k-fold cross-validation (Anthony and Holden, 1998). The special case of $k = N_{\mathcal{X}}$, it is also known as *leave-one-out* error and marked E_{LOO}. For ordinary least-squares regression it can be calculated analytically from the residual expansion coefficients and basis functions (Chapelle et al., 2002; Blatman and Sudret, 2010).

Given the sparse tree representation described in Section 1.2.2, it cannot be guaranteed that the residual $\mathcal{R}^{\ell}$ is expanded in every terminal domain. Therefore, during the partitioning phase of Algorithm 1 we initialise the error of all subdomains $\mathcal{D}_{\boldsymbol{X}}^{\ell+1,\{s_1,\cdots,s_{N_S}\}}$ of the current refinement domain $\mathcal{D}_{\boldsymbol{X}}^{\ell,p}$ to the leave-one-out error of the latter, $E_{\text{LOO}}^{\ell,p}$:

$$E_{\text{LOO}}^{\ell+1,\{s_1,\cdots,s_{N_S}\}} = E_{\text{LOO}}^{\ell,P}. \tag{1.27}$$

Then, we update the error estimate in each subdomain only if the conditions for its expansion hold (see Section 1.3.6.3). As a result, every terminal domain is either assigned its own leave-one-out error if it contains a residual expansion, or inherits the leave-one-out error from the last ancestor domain that was expanded.

By using the leave-one-out error in each terminal domain as an estimator of its generalisation error, i.e. $E_{\text{gen}}^{k} \approx \widehat{E}_{\text{LOO}}^{k}$, the empirical estimator of Eq. (1.26) that we call *weighted residual expansion error* reads:

$$E_{\text{WRE}} = \sum_{k\in\mathcal{T}} \widehat{E}_{\text{LOO}}^{k} \cdot \mathcal{V}^{k}. \tag{1.28}$$

To obtain a unit-less relative generalization error estimator, we normalize the weighed residual expansion error with the output variance to obtain

$$\epsilon_{\text{WRE}} = \frac{E_{\text{WRE}}}{\text{Var}\left[\mathcal{M}(\boldsymbol{X})\right]}. \tag{1.29}$$

An estimator for the output variance is either available as the sample variance from the training data in the case of a non-sequential experimental design (see Section 1.3.3), or analytically if the output variance is computed by post-processing the SSE (see Section 1.6.1).

1.6 Post-processing an SSE

1.6.1 Moments of an SSE

If PCE is used as a residual expansion technique, it becomes possible to analytically derive expressions for SSE output statistics. To this end, let us first introduce the notion of *flattened representation*: because PCE-based SSE is a polynomial in the original variables in every level and subdomain, this also holds for the terminal domains. Therefore, one can project the full

SSE in Eq. (1.11) as a local PCE onto each terminal domain:

$$\mathcal{M}^{F}_{\mathrm{SSE}}(\boldsymbol{X}^k) = \sum_{\boldsymbol{\alpha}\in\mathcal{A}^{\mathcal{T}}} c^p_{\boldsymbol{\alpha}} \Psi^k_{\boldsymbol{\alpha}}(\boldsymbol{X}^k), \tag{1.30}$$

where $\mathcal{A}^{\mathcal{T}}$ is a suitable truncation set for an exact projection, and the $c^p_{\boldsymbol{\alpha}}$ are the corresponding coefficients. The latter can easily be computed either analytically or exactly through quadrature. Note that, while the basis elements in the PCE in Eq. (1.30) correspond to the Ψ^k_j in Eq. (1.6) (they only depend on the input PDF in Eq. (1.8)), in general the coefficients will not be the same, i.e. $c^k_\alpha \neq a^k_\alpha$ in Eq. (1.6). From a technical perspective, the flattened representation in Eq. (1.30) contains all the information needed to evaluate Eq. (1.11) on a new point, but at a much lower storage cost, as only the final sets of coefficients $\boldsymbol{c}^p_{\boldsymbol{\alpha}}$ and basis indices $\mathcal{A}^{\mathcal{T}}$ need to be stored. This has additional advantages during the prediction of the response on new points, because it only requires the prediction of a single local expansion in the appropriate terminal domains, rather than that of all of its ancestors as in the original formulation in Eq. (1.11).

Because PCE contains as a basis element the constant term, it is straightforward to demonstrate that the expected value of Eq. (1.11) reads:

$$\mathbb{E}\left[\mathcal{M}_{\mathrm{SSE}}(\boldsymbol{X})\right] = \sum_{k\in\mathcal{T}} c^k_{\boldsymbol{0}}\, \mathcal{V}^k, \tag{1.31}$$

which is the weighted mean of all the mean values of the flattened representation in Eq. (1.30). Similarly, the variance can be calculated as:

$$\mathrm{Var}\left[\mathcal{M}_{\mathrm{SSE}}(\boldsymbol{X})\right] = \left(\sum_{k\in\mathcal{T}} \mathcal{V}^k \sum_{\boldsymbol{\alpha}\in\mathcal{A}^{\mathcal{T}}} \left(c^k_{\boldsymbol{\alpha}}\right)^2\right) - \mathbb{E}\left[\mathcal{M}_{\mathrm{SSE}}(\boldsymbol{X})\right]^2. \tag{1.32}$$

Both expressions Eqs. (1.31) and (1.32) hold for general, mutually dependent inputs if the basis functions $\Psi^k_{\boldsymbol{\alpha}}$ are orthogonal w.r.t. to the dependent input PDF. However, because of the way dependence is treated in the PCE-module of UQLAB (see UQLAB User Manual – Polynomial Chaos Expansions), this is not the case and output moments can only be computed for mutually independent inputs (i.e. $f_{\boldsymbol{X}}(\boldsymbol{x}) = \prod_{i=1}^{M} f_{X_i}(x_i)$) with the UQLAB SSE module.

1.6.2 Bayesian inversion

If SSE is used to approximate the likelihood function occurring in Bayesian inversion, many posterior quantities of interest can be computed analytically by post-processing the expansion coefficients. This way, SSE can be used as a sampling-free inversion approach. Refer to UQLAB User Manual – Bayesian inference for model calibration and inverse problems, Section 1.5 for details.

1.6.3 Reliability analysis

Modifying the sequential partitioning approach slightly, turns the SSE construction into an efficient active-learning reliability method. This is detailed in UQLAB User Manual – Structural Reliability, Section 1.6.2.

Chapter 2

Usage

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

2.1 Reference problem

We consider here the well-known *four-branch function* as a reference problem. This function is commonly used as a benchmark in the active learning reliability literature (Schueremans and Gemert, 2005; Echard et al., 2011). It simulates a series system with four distinct limit-state surfaces and is defined by the following set of equations:

$$\mathcal{M}(\boldsymbol{X}) = \min \begin{cases} 3 + 0.1(X_1 - X_2)^2 - \frac{X_1+X_2}{\sqrt{2}} \\ 3 + 0.1(X_1 - X_2)^2 + \frac{X_1+X_2}{\sqrt{2}} \\ X_1 - X_2 + \frac{6}{\sqrt{2}} \\ X_2 - X_1 + \frac{6}{\sqrt{2}}. \end{cases} \tag{2.1}$$

The input random vector is distributed according to a bivariate standard normal distribution, i.e. $\boldsymbol{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}_2)$.

From a surrogate modelling point of view, this function is challenging because it is non-smooth at the interface of the four limit-states. In this section we will approximate this model with SSE, making use of its local approximation strength.

2.2 Problem set-up

2.2.1 Initialize UQLAB

To setup the problem in UQLAB, we initialize UQLAB and fix a random seed for reproducibility:

```
uqlab
rng(100,'twister')
```

2.2.2 Defining the full model

The full model $\mathcal{M}$ described in Eq. (2.1) is implemented in the function `uq_fourbranch`, which we use to setup the full model as:

```
mOpts.mFile = 'uq_fourbranch';
myModel = uq_createModel(mOpts);
```

Further options for setting up models can be found in UQLAB User Manual – the MODEL module.

2.2.3 Defining the input distribution

The function `uq_fourbranch` accepts two inputs that are distributed according to the bivariate standard normal distribution. They can be added to UQLAB as follows:

```
for ii = 1:2
    iOpts.Marginals(ii).Type = 'Gaussian';
    iOpts.Marginals(ii).Parameters = [0 1];
end
myInput = uq_createInput(iOpts);
```

Additional options for specifying inputs can be found in UQLAB User Manual – the INPUT module.

2.3 Constructing the SSE metamodel

2.3.1 Initializing the metamodel

To construct an SSE metamodel in UQLAB, we create a `metaOpts` structure with the following fields:

```
metaOpts.Type = 'metamodel';
metaOpts.MetaType = 'SSE';
metaOpts.FullModel = myModel;
```

2.3.2 Analytical output moment computation

The first two output moments are available exactly if the input marginals are independent (see Section 1.6.1). However, the analytical computation requires the prior computation of the flattened SSE representation (see Section 2.8). This representation is not computed by default, because the flattening process requires additional computational resources that, for large polynomial degrees and large numbers of residual expansions, can become significant.

The output moment computation can be turned on explicitly with

```
metaOpts.PostProcessing.OutputMoments = true;
```

2.3.3 Experimental design

The experimental design used to construct the SSE metamodel is defined in the structure `metaOpts.ExpDesign`. For this reference problem, we let UQLAB sample an experimental design of size $N_{\mathcal{X}} = 200$ with

```
metaOpts.ExpDesign.NSamples = 200;
```

2.3.4 Creating the metamodel

After specifying the structure `metaOpts`, the SSE metamodel can be created with

```
mySSE = uq_createModel(metaOpts);
```

which returns

```
Warning: Flattened representation necessary for requested
         output moments, turning it on!

---           Calculating the SSE metamodel...           ---
Building baseline PCE...
Refinement step 1...
Refinement step 2...
Refinement step 3...
...
Refinement step 18...
Terminating algorithm, no more refinement domains found...
---                Calculation finished!                 ---
```

2.4 Assessing the metamodel

After constructing the metamodel `mySSE`, a summarizing report of the metamodel can be generated with

```
uq_print(mySSE)
```

which produces the following output

```
%-------- Stochastic spectral embedding output ---------%
   # input variables:                                   2
   # expansions:                                       32
   # levels:                                            6
   # refinement steps:                                 19
   # full model evaluations:                          200

   Error estimates
      Abs. weighted expansion error:          0.0055
      Rel. weighted expansion error:
         With sample variance:                0.0128
         With SSE variance:                    0.014
%-------------------------------------------------------%
```

Besides general information about the constructed SSE, this command displays information about the available error estimates discussed in Section 1.5. By default, the absolute weighted expansion error (see Eq. (1.28)) is displayed. If an estimate for the output variance is available, the relative weighted expansion error (see Eq. (1.29)) is also displayed. These estimates are either the output sample variance $\mathrm{Var}[\mathcal{Y}]$ (if non-sequential or user-specified experimental designs are used as discussed in Section 2.9.3) or the analytical SSE variance (if it is computed as discussed in Section 2.3.2).

A visual inspection of the generated model can be accessed with

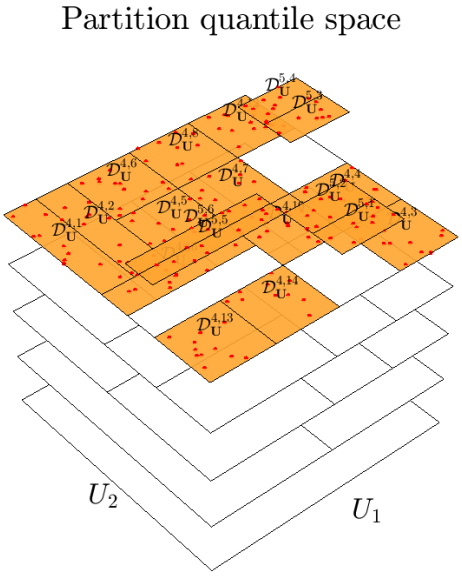
```
uq_display(mySSE)
```

This command generates a set of plots that help to understand the constructed SSE model. The plots generated for this reference problem are shown in Figure 6 (partitioning plots), Figure 7 (subdomain graph).

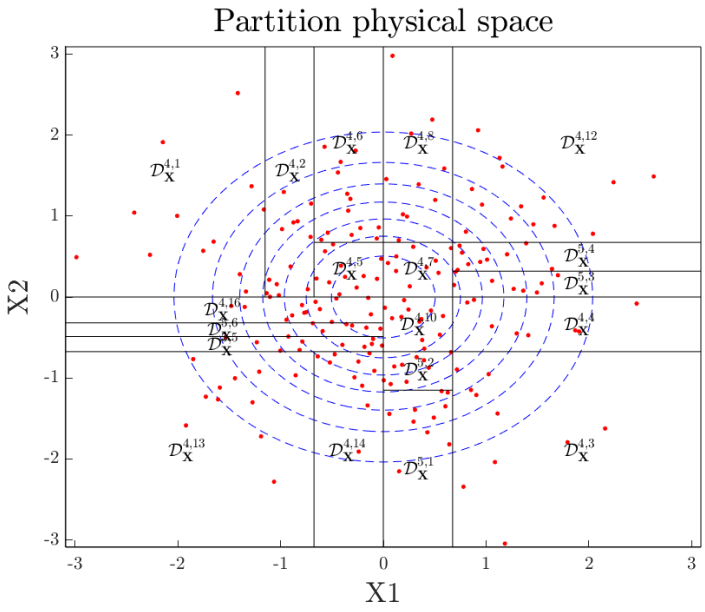
2.4.1 Use of validation set

If a validation set is provided (see Section 3.1.2), UQLAB automatically computes the validation error given in Eq. (1.23). To provide a validation set, the following command may be used:

```
metaOpts.ValidationSet.X = XVal;
metaOpts.ValidationSet.Y = YVal;
```



(a) $\mathcal{D}_{\boldsymbol{U}}$



(b) $\mathcal{D}_{\boldsymbol{X}}$

Figure 6: *Reference problem*: Partitions of the reference problem in the quantile space $\mathcal{D}_{\boldsymbol{U}}$ and physical space $\mathcal{D}_{\boldsymbol{X}}$ with the experimental design $\mathcal{X}$ shown as red points. The quantile space displays the partitions in 3D showing also non-terminal domains in white with the terminal domains $\mathcal{T}$ highlighted in orange. The physical space partition plot is a 2D plot including the input PDF $f_{\boldsymbol{X}}$ isolines but showing only the terminal domain partitions.

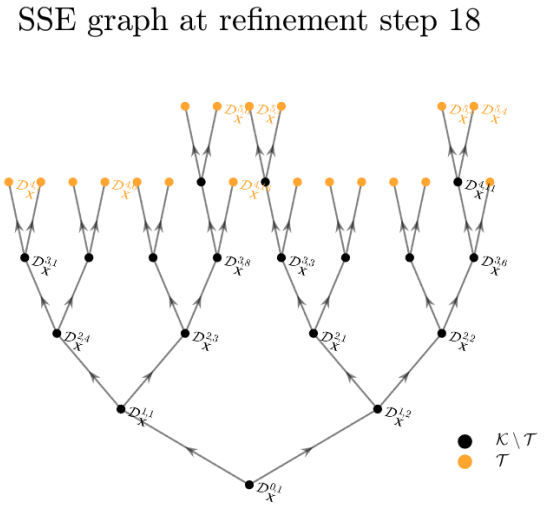


Figure 7: *Reference problem*: Graphical representation of the graph that stores the SSE structure. Terminal domains are denoted by orange points, all other domains by black points. The ancestry relationship between domains is denoted by the direction of arrows that point from *parent* nodes to *child* nodes.

The value of the validation error is stored in `mySSE.Error.Val` next to the other error measures (see Table 19) and will also be displayed when typing `uq_print(mySSE)`.

2.4.2 Additional display functionality

It is possible to display additional SSE plots by calling the `uq_display` function with optional arguments.

A plot showing the refinement scores at the last refinement step can be generated with

```
uq_display(mySSE,'refineScore',inf)
```

where `'refineScore'` and `inf` are a name-value pair. The `refineScore` value (`inf` in this example) can be set to any positive integer to show the refine scores at the i-th refinement step. The command generates the plot shown in Figure 8.

Another insightful plot can be generated with

```
uq_display(mySSE,'errorEvolution',true)
```

which displays the error evolution shown in Figure 9.

Figure 8: *Reference problem*: Refinement scores at the last refinement step of the algorithm. At the next step of the algorithm (see Algorithm 1), the refinement score is used to pick a refinement domain. The group *Others* gathers all domains that have a relative refinement score below 1%.

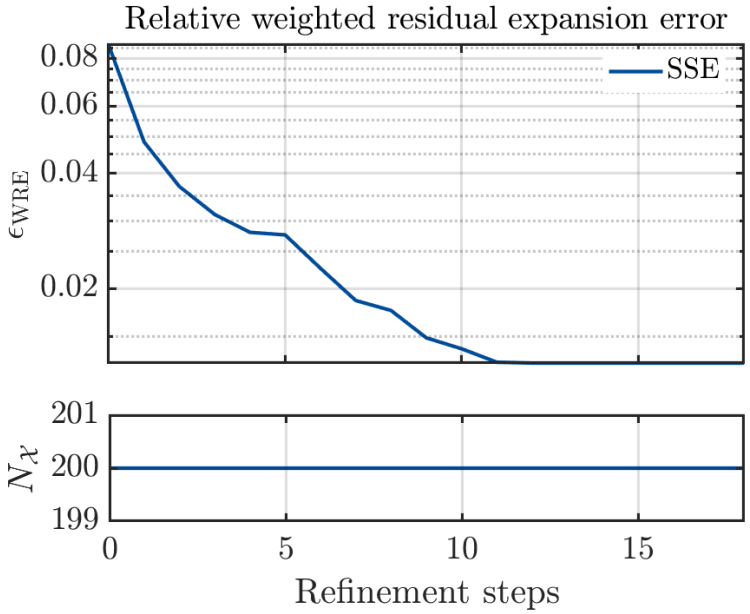


Figure 9: *Reference problem*: The upper subplot shows the evolution of the weighted residual expansion error calculated with Eq. (1.26). If neither a sequential nor a user-generated experimental design was used, the plot shows the relative error estimate that is normalized with the sample-based output variance (see Eq. (1.29)). Otherwise, the absolute error estimate is displayed (see Eq. (1.28)). The lower panel shows the evolution of the experimental design size $N_{\mathcal{X}}$.

2.5 Evaluating the metamodel

The constructed surrogate model can be evaluated on a new input sample with:

```
X = uq_getSample(1000);
Y = uq_evalModel(mySSE, X);
```

The `uq_evalModel` command accepts additional name-value pair arguments that, for instance, allow evaluating the SSE only up to a maximum refinement step.

2.6 Vector-valued models

If the model (or the experimental design, if manually specified) is vector-valued, i.e.$N_{\text{out}} > 1$, UQLAB constructs an independent SSE for each output component on the shared experimental design. No additional configuration is needed to enable this behaviour. Vector-valued models are not supported with sequential experimental designs (see Section 2.9.3).

An SSE with vector-valued outputs can be found in the UQLAB examples in:
`Examples/SSE/uq_Example_SSE_04_MultipleOutputs.m`.

Constructing an SSE on a vector-valued output model will result in a vector-valued output structure. As an example, a model with five outputs will produce the following structure:

```
mySSE.SSE

ans =

  1x5 sseClass array with fields:

    Graph
    FlatGraph
    Moments
```

Each element of the `mySSE` structure is functionally identical to its scalar counterpart in Table 14. Similarly, the `mySSE.Error` structure becomes a multi-element structure array:

```
mySSE.Error

ans =

  1x5 struct array with fields:

    normEmpErr
    AbsWRE
    RelWRE
```

2.7 Constant parameters

In some analyses, one may need to assign a constant value to one or to a set of parameters. When this is the case, the SSE metamodel is built by internally removing the constant parameters from the inputs. This process is transparent to the users as they shall still evaluate the model using the full set of parameters (including those which were set constant). UQLAB will automatically and appropriately account for the set of input parameters which were declared constant.

As an example, assume that the first input of the four-branch reference problem is set to $x_1 = 5$ with

```
iOpts.Marginals(1).Type = 'Constant';
iOpts.Marginals(1).Parameters = 0.5;
myInput = uq_createInput(iOpts);
```

The SSE metamodel can then be constructed identically to Section 2.3.4.

When evaluating the metamodel, the constant parameter has to be included in the parameter set:

```
uq_evalModel(mySSE, [0 0.5])
```

2.8 Flattened representation

The flattened representation of an SSE metamodel (see Eq. (1.30)) can be computed to reduce the computational resources required to store and evaluate an SSE metamodel. This representation is also required to analytically compute the first two output moments of a constructed SSE metamodel as discussed in Section 2.3.2.

To compute the flattened representation of an SSE, the following option may be set in the `metaOpts` structure:

```
metaOpts.PostProcessing.Flatten = true;
```

2.9 Advanced options for SSE metamodel

The SSE module of UQLAB allows for the configuration of all steps in the sequential partitioning algorithm introduced in Algorithm 1.

2.9.1 Refinement domain selection

The refinement domain selection, as discussed in Section 1.3.1, depends on a refinement score that is updated at every iteration of the algorithm. By default, the refinement score is computed with the function `uq_SSE_refineScore_LOO`, that implements the refinement score computation based on the weighted leave-one-out error of the residual expansion (see Section 1.3.1).

To specify any other custom refinement score, the following options can be passed to the `metaOpts` structure

```
metaOpts.Refine.Score = ...
   @(obj, subIdx) uq_SSE_refineScore(obj, subIdx);
```

Here, `uq_SSE_refineScore` can be any function that accepts as an argument the SSE object `obj` and the index of the subdomain `subIdx` for which the refinement score is computed. The function returns the refinement score, i.e.a scalar for each domain that is used to select a refinement domain (see Eq. (1.13)). For more details, refer to Table 11.

The set of refinement domains $\tilde{\mathcal{T}} \subseteq \mathcal{T}$ considered for refinement can be constructed with two criteria that are both considered equivalently. The first, is to introduce a refinement score threshold τ_{ref} with

```
metaOpts.Refine.Threshold = 1e-3;
```

When this threshold is set, only terminal domains with a relative refinement score above 10^{-3} will be considered for refinement (see Eq. (1.16)).

Additionally, it is possible to construct expansions only domains that contain at least a fixed number of refinement points N_{exp}. This can be set to a desired value (e.g.$N_{\text{exp}} = 5$) with:

```
metaOpts.Refine.NExp = 5;
```

If the experimental design is sampled sequentially (see Section 2.9.3), the enrichment of the experimental design happens before checking this condition.

Specifying `.NExp` also acts as an indirect stopping criterion, because it limits the number of domains in which an expansion can be constructed (see Section 1.3.6.3).

2.9.2 Partitioning strategy

The partitioning strategy determines how a selected refinement domain is partitioned into a number of subdomains. By default, the partitioning is carried out by the `uq_SSE_partitioning_Sobol` function, that partitions according to the estimated first order Sobol' index in the subdomain (see Section 1.3.2.1).

To specify other custom partitioning strategies, the following options can be passed to the `metaOpts` structure

```
metaOpts.Partitioning.Strategy = ...
   @(obj, subIdx) uq_SSE_partitioning(obj, subIdx);
```

Here, `uq_SSE_partitioning` can be any function that accepts as an argument the SSE object `obj` and the index of the subdomain `subIdx` that is partitioned. The function returns a structure array `newDomains` (see Table 9) that characterizes the subdomains after partitioning. In principle, an arbitrary number of subdomains can be introduced, as long as they form a complete partition of the refinement domain. For more details, refer to Table 8.

2.9.3 Sequential experimental design

Instead of sampling the full experimental design before starting the construction of the metamodel, SSE also supports an option to sequentially enrich the experimental design during the SSE construction. If this is desired, the following option can be passed to `metaOpts` structure

```
metaOpts.ExpDesign.Sampling = 'Sequential';
metaOpts.ExpDesign.NSamples = 200;
```

Note: Sequential sampling of the experimental design is only supported for scalar outputs, i.e. $N_{\text{out}} = 1$

The sequential design will then be enriched sequentially until `.NSamples` have been added. With this option, the enrichment takes place at every iteration of the sequential partitioning algorithm after new domains have been created by partitioning the selected refinement domain (see Algorithm 1). To specify that 5 points should be added at every refinement step, the `.NEnrich` field can be set with

```
metaOpts.ExpDesign.NEnrich = 5;
```

By default, the enrichment strategy is defined in the function `uq_SSE_enrichment_uniform`. For even more control on how new sample points are added to the experimental design, a custom enrichment strategy may be defined with

```
metaOpts.ExpDesign.Enrichment = @(obj, currBounds, NEnrich) ...
   uq_SSE_enrichment(obj, bounds, NEnrich);
```

Here, `uq_SSE_enrichment` can be any function that accepts as an argument the SSE object `obj`, the bounds in the quantile space `bounds` where the experimental design is to be enriched, and the number of points to be added to the experimental design `NEnrich`. The function returns the newly added sample points in the physical space and the quantile space. For more details, refer to Table 10.

2.9.4 Residual expansion

The SSE module currently only supports the powerful UQLAB PCE module (UQLAB User Manual – Polynomial Chaos Expansions) to construct the local residual expansions $\widehat{\mathcal{R}}_{\mathrm{S}}$ (see Section 1.3.4). In principle, any spectral metamodelling technique may be used instead of PCE. PCE, however, has attractive properties that allow post-processing PCE-based SSE and computing various quantities of interest at no additional cost in terms of full model evaluations (see Section 1.6).

To specify the residual expansion properties, the `metaOpts.ExpOptions` field accepts a structure defining the residual expansion properties (see the `MetaOpts` structure in UQLAB User Manual – Polynomial Chaos Expansions). As an example, to use PCEs with with degree- and q-norm-adaptivity, the structure can be set to

```
metaOpts.ExpOptions.MetaType = 'PCE';
metaOpts.ExpOptions.Degree = 0:4;
metaOpts.ExpOptions.TruncOptions.qNorm = 0.5:0.1:0.8;
```

Note: It is possible to control all aspects of the PCEs by passing expansion-specific options (see `MetaOpts` structure in UQLAB User Manual – Polynomial Chaos Expansions) to `metaOpts.ExpOptions`.

2.9.5 Post-expansion operation

To provide an additional degree of flexibility when constructing the SSE surrogate model, the SSE module allows the user to specify a *post-expansion* operation that is executed after the residual expansion is constructed (see Algorithm 1). It can be specified with

```
metaOpts.PostExpansion = ...
   @(obj, subIdx) uq_SSE_postExpansion(obj, subIdx);
```

This operation is a user-defined function that accepts the full SSE object `obj` and the indices of the subdomains created in the current refinement step `subIdx`. It returns the nodes stored in the SSE graph structure. Therefore, this function gives the user complete freedom in modifying the nodes structure after every expansion, for example, to add custom properties to the graph node such as the conditional failure probabilities (see UQLAB User Manual – Structural Reliability, Section 1.6.2). For more details, refer to Table 7.

2.9.6 Stopping criterion

As discussed in Section 1.3.6, different criteria are checked to stop the algorithm. First and foremost, the algorithm checks whether the refinement set is empty, i.e. $\tilde{\mathcal{T}} = \emptyset$, in which case

the algorithm is immediately stopped.

Additionally, the algorithm is terminated once a fixed number of refinements have been undertaken. The number of allowed refinements can be modified with

```
metaOpts.Stopping.maxRefine = 100;
```

which sets the number of allowed refinement steps to 100.

Finally, it is also possible to specify a user-defined stopping criterion with

```
metaOpts.Stopping.Criterion = @(obj) uq_SSE_stopping(obj);
```

Here, `uq_SSE_stopping` can be any function that accepts as an argument the SSE object `obj` and returns a `true` if the algorithm is to be stopped and `false` otherwise. For more details, refer to Table 12.

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 the SSE metamodel

Syntax

```
mySSE = uq_createModel(Metaopts)
```

Input

The struct variable `Metaopts` contains the configuration information for a stochastic spectral embedding metamodel. The detailed list of available options is reported in Table 1.

Table 1: `Metaopts`

●	`.Type`	`'Metamodel'`	Select the metamodeling tool
●	`.MetaType`	`'SSE'`	Select stochastic spectral embedding
☐	`.Name`	String	Unique identifier for the metamodel
☐	`.Display`	String default: `'standard'`	Level of information displayed by the methods
		`'quiet'`	Minimum display level, displays nothing or very few information.
		`'standard'`	Default display level, shows the most important information.
		`'verbose'`	Maximum display level, shows all the information on runtime, like updates on iterations, etc.
●	`.ExpDesign`	Table 2	Experimental design options
☐	`.ValidationSet`	Table 3	Validation set (Section 1.23)
☐	`.Input`	String	The name of the INPUT object that describes the inputs of the metamodel
		INPUT object	The INPUT object that describes the inputs of the metamodel
☐	`.FullModel`	String	The name of the MODEL object that is used to calculate the model responses
		MODEL object	The MODEL object that is used to calculate the model responses
☐	`.Refine`	Table 4	Refinement options
☐	`.PostExpansion`	Function handle	Action carried out directly after the expansion. See Table 7 for details.

☐	`.Partitioning`	Function handle	Partitioning strategy. See Table 8 for details.

☐	`.ExpOptions`	Structure
		default:
		
		`.Type ='Metamodel'`
		`.MetaType ='PCE'`
		`.Method ='LARS'`
		`.Degree = 0:4`
		`.Display = 0`
			Options for the residual expansion. See UQLAB User Manual – Polynomial Chaos Expansions.

☐	`.PostProcessing`	Table 5	Post-processing options.
☐	`.Stopping`	Table 6	Options for stopping the algorithm.

3.1.1 Experimental design options

The experimental design $\mathcal{X}$ and corresponding model evaluations $\mathcal{M}(\mathcal{X})$ are used to construct the SSE. Table 2 lists the available options.

Table 2: `Metaopts.ExpDesign`

⊞	`.Sampling`	String default: `'MC'`	Sampling type
		`'MC'`	Monte Carlo sampling.
		`'LHS'`	Latin Hypercube sampling.
		`'Sobol'`	Sobol sequence sampling.
		`'Halton'`	Halton sequence sampling.
		`'Sequential'`	Sequential sample enrichment.
⊞	`.DataFile`	String	A string containing the name of the mat file that contains the experimental design.
⊞	`.X`	$N \times M$ Double	User defined model input $\mathcal{X}$.
⊞	`.Y`	$N \times O$ Double	User defined model response $\mathcal{M}(\mathcal{X})$.
☐	`.NSamples`	Integer	The number of samples to draw.
☐	`.NEnrich`	Integer	The number of samples to draw.
☐	`.Enrichment`	Function handle	Enrichment strategy. See Table 10 for details.

3.1.2 Validation Set

If a validation set is provided, UQLAB automatically calculates the validation error of the created SSE. The required information is listed in Table 3.

Table 3: `Metaopts.ValidationSet`			
●	`.X`	$N \times M$ Double	User-specified validation set $\mathcal{X}_{Val}$
●	`.Y`	$N \times N_{Out}$ Double	User-specified validation set response $\mathcal{M}(\mathcal{X})$

3.1.3 Refinement options

The refinement options specify how the sequential partitioning algorithm selects refinement domains. All options are listed in Table 4.

Table 4: `Metaopts.Refine`			
☐	`.NExp`	Integer default: 10	Minimum number of sample points N_{exp} required to construct a residual expansion.
☐	`.Threshold`	Double default: -inf	For refinement, consider only subdomains that have a relative refinement score (see Eq. (1.16)) above the specified threshold τ_{ref}.
☐	`.Score`	Function handle	Handle to function that returns the refinement score. See Table 11 for details.

3.1.4 Post-processing options

After constructing an SSE, it is possible to post-process it with the options listed in Table 5.

Table 5: `Metaopts.PostProcessing`			
☐	`.Flatten`	Boolean default: false	Switch to flatten the SSE after construction (Section 1.6.1).
☐	`.OutputMoments`	Boolean default: false	Switch to compute output moments.

3.1.5 Stopping criterion

In order to terminate the algorithm, different stopping criteria can be specified as listed in Table 6.

Table 6: `Metaopts.Stopping`

☐	`.MaxRefine`	Integer default: 1000	Maximum number of refinement steps
☐	`.Criterion`	Function handle	Stopping criterion. See Table 12 for details.

3.2 Custom function handle specifications

Different function handles are used to specify the behaviour of the sequential partitioning algorithm.

3.2.1 `uq_SSE_postExpansion`

This function lets users specify a post-expansion operation as discussed in Section 2.9.5.

Table 7: `nodes = uq_SSE_postExpansion(obj,subIdx)`		
`obj`	SSE	SSE object.
`subIdx`	Integer	Which subdomain the post-expansion action pertains to.
`nodes`	Table	The modified `obj.graph.nodes`

3.2.2 `uq_SSE_partitioning`

This function lets users specify a partitioning strategy as discussed in Section 2.9.2.

Table 8: `newDomains = uq_SSE_partitioning(obj,subIdx)`		
`newDomains`	Table 9	The partition of the `subIdx` domain.
`obj`	SSE	SSE object.
`subIdx`	Integer	Which subdomain is partitioned.

Table 9: `newDomains(ii)`		
`.bounds`	$2 \times M$ double	Bounds of the `ii`-th new domain in the quantile space.
`.inputMass`	Double	Probability mass of the `ii`-th new domain.

3.2.3 uq_SSE_enrichment

This function lets users specify an enrichment strategy as discussed in Section 2.9.3.

Table 10: `[U_new, X_new] = uq_SSE_enrichment(obj,bounds,NEnrich)`		
`U_new`	`NEnrich`$\times M$	Newly added sample points in quantile space.
`X_new`	`NEnrich`$\times M$	Newly added sample points in physical space.
`obj`	SSE	SSE object.
`bounds`	$2 \times M$ double	Bounds in quantile space in which to enrich.
`NEnrich`	Integer	Number of sample points to add.

3.2.4 uq_SSE_refineScore

This function lets users specify a custom refinement score to modify the refinement domain selection as discussed in Section 2.9.1.

Table 11: `refineScore = uq_SSE_refineScore(obj,subIdx)`		
`refineScore`	`Double`	The refinement score. The domain with the largest refinement score is refined in the next refinement step.
`obj`	SSE	SSE object.
`subIdx`	Integer	For which domain the refinement score is returned.

3.2.5 uq_SSE_stopping

This function lets users specify a custom stopping criterion score to prematurely terminate the sequential partitioning algorithm as discussed in Section 2.9.6.

Table 12: `stop = uq_SSE_stopping(obj)`		
`stop`	`Boolean`	Is the stopping criterion met? If `stop` is true, refinement is stopped after the current step, if `stop` is false, refinement is continued.
`obj`	SSE	SSE object.

3.3 Accessing the results

Syntax

```
mySSE = uq_createModel(MetaOpts) ;
```

Output

Regardless on the configuration options given at creation time in the `MetaOpts` structure, all SSE metamodels share the same output structure, given in Table 13.

Table 13: `mySSE`		
`.Name`	String	Unique name of the SSE metamodel.
`.Options`	Table 1	Copy of the `Metaopts` structure used to create the metamodel including default options.
`.SSE`	Table 14	Information about the final SSE model.
`.ExpDesign`	Table 18	Experimental design used to construct the SSE.
`.Error`	Table 19	Error estimates of the metamodel accuracy.

Table 14: `mySSE.SSE`		
`.Graph`	Table 15	The graph structure used to store the SSE.
`.FlatGraph`	Table 15	A flattened version of the `.Graph` object.
`.Moments`	Table 17	A structure storing the analytically computed first two SSE moments. Is non-empty only if moment computation was turned on explicitly (see Section 2.3.2).

Table 15: `mySSE.SSE.Graph`		
`.Nodes`	Table 16	The main storage of the SSE object. Every node corresponds to one subdomain in the SSE structure, containing all relevant information.
`.Edges`	Table	Stores information about the connection between nodes.

Table 16: `mySSE.SSE.Graph.Nodes`		
`.neighbours`	Integer vector	The indices of the neighbouring domains, i.e. domains that share a common boundary.
`.bounds`	$2 \times M$ double	Bounds of the domains in the quantile space.
`.inputMass`	Double	The probability mass of the domain.

.ref	Integer	Refinement step at which the subdomain was created.
.level	Integer	Level of the subdomain.
.idx	Integer	Level-wise index of the subdomain.
.expansions	MODEL	The expansion object in the subdomain. If field is empty, no expansion was created in the subdomain.
.refineScore	Double	Value of the refinement score computed with uq_SSE_refineScore.

Table 17: mySSE.SSE.Moments

.mean	Double	The analytically computed SSE output mean (see Eq. (1.31)).
.var	Double	The analytically computed SSE output variance (see Eq. (1.32)).

Table 18: mySSE.ExpDesign

.NSamples	Integer	The total number of samples.
.Sampling	String	The sampling method.
.ED_Input	INPUT object	The input module that was used to generate the experimental design.
.Enrichment	Function handle	Enrichment function handle.
.NEnrich	Integer	The number of samples added at each refinement step.
.X	$N \times M$ Double	The experimental design values $\mathcal{X}$.
.U	$N \times M$ Double	The experimental design values in the quantile space.
.Y	$N \times N_{\text{out}}$ Double	The output $\mathcal{Y}$ of the model that corresponds to the input $\mathcal{X}$.
.Res	$N \times N_{\text{out}}$ Double	The residual between $\mathcal{Y}$ and $\mathcal{M}_{\text{SSE}}(\mathcal{X})$.
.ref	$N \times N_{\text{out}}$ Integer	The refinement step at which the sample point was generated.

Table 19: mySSE.Error

.AbsWRE	Double	The absolute (i.e. not-normalized) weighted residual expansion error E_{WRE} (see Eq. (1.28)).

`.RelWRE`	Double	The relative (i.e. normalized) weighted residual expansion error ϵ_{WRE} (see Eq. (1.29)). Only available if a variance estimator is available.
`.normEmpErr`	Double	The relative (i.e. normalized) empirical error ϵ_{emp} (see Eq. (1.24)).
`.Val`	Double	Validation error ϵ_{val} (see Eq. (1.23). Only available if a validation set is provided (see Table 3).

Note: In general the fields `mySSE.SSE` and `mySSE.Error` are structure arrays with length equal to the number of outputs N_{out} of the metamodel.

3.4 Evaluating the metamodel (predictor)

Syntax

```
Y = uq_evalModel(X)
Y = uq_evalModel(mySSE, X)
Y = uq_evalModel(..., NAME, VALUE)
```

Description

`Y = uq_evalModel(X)` returns the value of the SSE prediction ($N \times N_{\text{out}}$ Double) on the points `X` ($N \times M$ Double) using the last created SSE model.

Note: by default, the *last created* model or surrogate model is the currently active model.

`Y = uq_evalModel(mySSE,X)` returns the value of the SSE prediction ($N \times N_{\text{out}}$ Double) on the points `X` ($N \times M$ Double) using the SSE model object `mySSE`.

`Y = uq_evalModel(..., NAME, VALUE)` specifies additional options for the SSE evaluation given as name-value pairs listed in Table 20.

Table 20: `mySSE`

`'maxRefine'`	Integer default: Inf	Number of refinement steps considered in the evaluation.
`'varDim'`	Integer vector default: `[1:M]`	Input dimensions to consider in the evaluation. Excluded dimensions are marginalized.

References

Anthony, M. and Holden, S. B. (1998). Cross-validation for binary classification by real-valued functions. In *Proceedings of the eleventh annual conference on Computational learning theory - COLT' 98*. ACM Press. 13

Blatman, G. and Sudret, B. (2010). An adaptive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis. 25:183–197. 13

Breiman, L. (2017). *Classification and regression trees*. Routledge. 4

Chapelle, O., Vapnik, V., and Bengio, Y. (2002). Model selection for small sample regression. 48(1):9–23. 13

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

Friedman, J. H. (1991). Multivariate adaptive regression splines. 19(1):1–67. 4

Hrinda, G. (2010). Snap-through instability patterns in truss structures. In *51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference 18th AIAA/ASME/AHS Adaptive Structures Conference 12th*, page 2611. 1

Marelli, S., Wagner, P.-R., Lataniotis, C., and Sudret, B. (2021). Stochastic spectral embedding. 11(2):25–47. 1

Nagel, J. B. and Sudret, B. (2016). Spectral likelihood expansions for Bayesian inference. 309:267–294. 1

Rosenblatt, M. (1952). Remarks on a multivariate transformation. 23(3):470–472. 4

Schueremans, L. and Gemert, D. V. (2005). Benefit of splines and neural networks in simulation based structural reliability analysis. *Struct. Saf.*, 27(3):246–261. 17

Serna, A. and Bucher, C. (2009). Advanced surrogate models for multidisciplinary design optimization. In *8th Weimar Optimization and Stochastic Days*. 1

Shields, M. D. (2018). Adaptive Monte Carlo analysis for strongly nonlinear stochastic systems. 175:207–224. 8

Sobol', I. M. (1993). Sensitivity estimates for nonlinear mathematical models. 1(4):407–414. 8

Sudret, B. (2008). Global sensitivity analysis using polynomial chaos expansions. 93(7):964–979. 8

Torre, E., Marelli, S., Embrechts, P., and Sudret, B. (2019a). Data-driven polynomial chaos expansion for machine learning regression. *Journal of Computational Physics*, 388:601–623. 5

Torre, E., Marelli, S., Embrechts, P., and Sudret, B. (2019b). A general framework for data-driven uncertainty quantification under complex input dependencies using vine copulas. 55:1–16. 4