ETH zürich

UQLab User Manual
Canonical Low-rank Approximations

K. Konakli, C. Mylonas, 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

K. Konakli, C. Mylonas, S. Marelli, B. Sudret, UQLab user manual – Canonical low-rank approximations, Report UQLab-V2.1-108, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland, 2024

BibTeX entry

```
@TechReport{UQdoc_21_108,
author = {Konakli, K. and Mylonas, C. and Marelli, S. and Sudret, B.},
title = {{UQLab user manual -- Canonical low-rank approximations}},
institution = {Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich,
Switzerland},
year = {2024},
note = {Report UQLab-V2.1-108}
}
```

Document Data Sheet

Document Ref.	UQLAB-V2.1-108
Title:	UQLAB user manual – Canonical low-rank approximations
Authors:	K. Konakli, C. Mylonas, S. Marelli, B. Sudret Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland
Date:	15/04/2024

Doc. Version	Date	Comments
V2.1	15/04/2024	UQLAB V2.1 release
V2.0	01/02/2022	UQLAB V2.0 release
V1.4	01/02/2021	UQLAB V1.4 release
V1.3	19/09/2019	UQLAB V1.3 release • Minor updates to the plots.
V1.2	22/02/2019	UQLAB V1.2 release • added automatic calculation of validation error if a validation set is provided
V1.1	05/07/2018	UQLAB V1.1 release
V1.0	01/05/2017	Initial release

Abstract

Metamodeling techniques are gaining increasing popularity as a powerful approach for propagating uncertainties through complex computational models. Canonical low-rank approximations (LRA), also known as separated representations, have recently been introduced in this context as a promising tool for effectively dealing with cases of high-dimensional model input. The key idea thereof is to approximate a response quantity of interest with a sum of a small number of appropriate rank-one tensors, which are products of univariate functions. The rank-one tensors are herein built with polynomial functions.
The UQLAB LRA module allows one to easily build and deploy canonical LRA metamodels non-intrusively. The manual is divided into the following three parts:

- A brief introduction to the formulation, construction and post-processing of canonical LRA with polynomial basis;
- A detailed illustration of the module usage and the available options through the step-by-step unfolding of a comprehensive example;
- A reference list summarizing the available functionalities of the module.

Keywords: UQLAB, metamodeling, canonical low-rank approximations, separated representations, polynomial basis

Contents

Chapter 1

Theory

1.1 Introduction

Propagation of uncertainties through complex computational models often becomes practically infeasible due to the large number of time-consuming simulations required. Metamodelling offers a remedy to this limitation by substituting a complex model with a statistically equivalent one that is inexpensive to evaluate; the latter is the so-called metamodel. The present manual documents the use of UQLAB for creating metamodels belonging to the particular class of canonical low-rank approximations (LRA).

We note that canonical LRA built with polynomial functions offer an alternative metamodelling technique to the more popular polynomial chaos expansions (PCE). In the latter, the model response is expanded onto a basis of orthonormal multivariate polynomials, which are obtained as tensor products of univariate polynomials in the input parameters (see also UQLAB User Manual – Polynomial Chaos Expansions). On the other hand, LRA exploit the tensor-product form of the multivariate basis, leading to a drastic reduction of the number of unknown coefficients. We emphasize that the number of unknowns in LRA grows *only linearly* with the input dimension, which makes the approach particularly promising for high-dimensional problems.

The current chapter describes the formulation of canonical LRA with polynomial functions and details an algorithm for their computation. Moreover, it presents the post-processing of the LRA coefficients for computing the mean and variance of the model response *analytically*.

1.2 Canonical low-rank approximations with polynomial basis

1.2.1 General formulation

We consider a physical or engineering system whose behavior is represented by a computational model $\mathcal{M}$. Let $\boldsymbol{X} = \{X_1, \ldots, X_M\}$ denote the M-dimensional input random vector with joint probability density function (PDF) $f_{\boldsymbol{X}}$ and associated marginals f_{X_i}, $i = 1, \ldots, M$. We herein assume that the components of $\boldsymbol{X}$ are *independent* and will address the case of dependent input in a subsequent section. We further consider a scalar response quantity of

interest denoted by Y. The above are summarized in the mathematical formalism:

$$\boldsymbol{X} \in \mathcal{D}_{\boldsymbol{X}} \subset \mathbb{R}^M \longmapsto Y = \mathcal{M}(\boldsymbol{X}) \in \mathbb{R}, \tag{1.1}$$

where $\mathcal{D}_{\boldsymbol{X}}$ denotes the support of $\boldsymbol{X}$.
Employing the definition of *canonical* rank, a rank-one function of $\boldsymbol{X}$ has the form:

$$w(\boldsymbol{X}) = \prod_{i=1}^{M} v^{(i)}(X_i), \tag{1.2}$$

where $v^{(i)}$ denotes a univariate function in the i-th dimension. Accordingly, a rank-R approximation of $Y = \mathcal{M}(\boldsymbol{X})$ reads:

$$\widehat{Y}^{\mathrm{LRA}} = \widehat{\mathcal{M}}^{\mathrm{LRA}}(\boldsymbol{X}) = \sum_{l=1}^{R} b_l \left(\prod_{i=1}^{M} v_l^{(i)}(X_i) \right), \tag{1.3}$$

where $v_l^{(i)}$ denotes the i-th dimensional univariate function in the l-th rank-one component and $\{b_l,\ l = 1, \ldots, R\}$ are scalars that can be viewed as weighing factors. Assuming that the number R of rank-one terms is small, the right-hand side of Eq. (1.3) represents a canonical low-rank approximation (LRA). By expanding $v_l^{(i)}$ onto a polynomial basis that is orthonormal with respect to the marginal f_{X_i}, Eq. (1.3) takes the form:

$$\widehat{Y}^{\mathrm{LRA}} = \widehat{\mathcal{M}}^{\mathrm{LRA}}(\boldsymbol{X}) = \sum_{l=1}^{R} b_l \left(\prod_{i=1}^{M} \left(\sum_{k=0}^{p_i} z_{k,l}^{(i)}\, \phi_k^{(i)}(X_i) \right) \right), \tag{1.4}$$

where $\phi_k^{(i)}$ denotes the k-th degree univariate polynomial in the i-th input variable, p_i is the maximum degree of $\phi_k^{(i)}$ and $z_{k,l}^{(i)}$ is the coefficient of $\phi_k^{(i)}$ in the l-th rank-one component.

1.2.2 Comparison to polynomial chaos expansions

Polynomial chaos expansions (PCE) constitute a popular approach for developing metamodels with polynomial bases, which has been used widely in a variety of applications in the last two decades (Blatman and Sudret, 2011; Deman et al., 2016). It is thus of interest to provide a brief comparison of PCE to canonical LRA built with polynomial functions.
We consider again the map in Eq. (1.1) employing the same assumption of independence for the components of $\boldsymbol{X}$. A PCE approximation of $Y = \mathcal{M}(\boldsymbol{X})$ has the form (Xiu and Karniadakis, 2002; Soize and Ghanem, 2004):

$$Y^{\mathrm{PCE}} = \mathcal{M}^{\mathrm{PCE}}(\boldsymbol{X}) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}} y_{\boldsymbol{\alpha}}\, \Psi_{\boldsymbol{\alpha}}(\boldsymbol{X}), \tag{1.5}$$

where $\mathcal{A}$ is a set of multi-indices $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)$, $\{\Psi_{\boldsymbol{\alpha}},\ \boldsymbol{\alpha} \in \mathcal{A}\}$ is a set of multivariate polynomials that are orthonormal with respect to $f_{\boldsymbol{X}}$ and $\{y_{\boldsymbol{\alpha}},\ \boldsymbol{\alpha} \in \mathcal{A}\}$ is the set of polynomial coefficients. Due to the independence of the input variables, the orthonormal polynomial bases in Eq. (1.5) can be obtained by tensorization of univariate polynomials

that are orthonormal with respect to the marginals f_{X_i}:

$$\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X}) = \prod_{i=1}^{M} \phi_{\alpha_i}^{(i)}(X_i), \tag{1.6}$$

where $\phi_{\alpha_i}^{(i)}$ is a univariate polynomial of degree α_i in the i-th input variable. Obviously, the families of the univariate polynomials used to formulate the multivariate PCE basis are the same as the families of polynomials that form the bases of the univariate functions in LRA. However, as seen in Eq. (1.4), LRA retain the tensor-product form of Eq. (1.6), whereas the expanded form is considered in PCE.

The advantage of representing the spectral expansion in Eq. (1.5) in the form of a tensor decomposition is that it sets up a framework which allows solving for the coefficients *in each dimension separately* without explicitly carrying out the tensor product in Eq. (1.6). In this way, it facilitates uncertainty propagation through models with inputs of high dimension. It should be emphasized that a significant reduction in the number of unknowns is achieved while maintaining the higher-order effects, whereas such effects are often neglected in the PCE framework due to the required basis truncation (see UQLAB User Manual – Polynomial Chaos Expansions for further details on the basis truncation).

1.3 Construction of canonical low-rank approximations

Building a canonical LRA of a certain rank requires (i) the specification of the univariate polynomial basis along each dimension and (ii) the evaluation of the polynomial coefficients and weighing factors; these are respectively described in paragraphs 1.3.1 and 1.3.2. Note however that in a typical application of LRA, the optimal rank is not known *a priori*; an approach for rank selection based on metamodel error estimates is discussed in paragraph 1.3.3. Finally, paragraph 1.3.4 refers to the selection of the polynomial degree.

We underline that the metamodel is herein constructed non-intrusively, *i.e.* without any alteration of the original model $\mathcal{M}$. The non-intrusive construction relies on a set of samples from the uncertain input, comprising the *experimental design* $\mathcal{X}$, and the corresponding model responses $\mathcal{Y}$.

1.3.1 The polynomial basis

As noted in Section 1.2.1, the univariate polynomials in Eq. (1.4) are orthonormal with respect to the corresponding marginal distributions. Therefore, the polynomials in the i-th dimension satisfy the condition:

$$\left\langle \phi_j^{(i)}, \phi_k^{(i)} \right\rangle \overset{\text{def}}{=} \int_{\mathcal{D}_{X_i}} \phi_j^{(i)}(x_i)\phi_k^{(i)}(x_i) f_{X_i}(x_i)\mathrm{d}x_i = \delta_{jk} \tag{1.7}$$

where δ_{jk} is the Kronecker delta symbol. Note that the right-hand side of the above definition represents the expectation value of the product of the j-th and k-th degree polynomials.

Table 1 lists the classical families of univariate orthonormal polynomials together with the distributions to which they are orthonormal (Sudret, 2007). A detailed description of the polynomial families in the table can be found in *e.g.* Xiu and Karniadakis (2002). Cases when the input variables do not follow standard distributions can be treated through an isoprobabilistic transformation: first, the random vector $\boldsymbol{X}$ is transformed into a basic random vector $\boldsymbol{U}$, *i.e.* a vector of variables following standard distributions, by means of an isoprobabilistic transformation $\boldsymbol{U} = T(\boldsymbol{X})$. Then, the model response $\mathcal{M}(T^{-1}(\boldsymbol{U}))$ is expanded onto the polynomial basis associated with the basic vector $\boldsymbol{U}$. The same concept can be employed to deal with cases with mutually dependent input variables by transforming them into a vector of independent variables through an isoprobabilistic mapping Lebrun and Dutfoy (2009).

It is also possible to construct sets of univariate polynomials that are orthonormal with respect to an arbitrary probability distribution. This feature is available in UQLAB based on the Stieltjes procedure (Gautschi, 2004); for further details, please refer to UQLAB User Manual – Polynomial Chaos Expansions.

Table 1: List of classical univariate polynomial families and associated distributions.

Type of variable	Distribution	Orthogonal polynomials	Hilbertian basis $\psi_k(x)$
Uniform	$\mathbf{1}_{]-1,1[}(x)/2$	Legendre $P_k(x)$	$P_k(x)/\sqrt{\frac{1}{2k+1}}$
Gaussian	$\frac{1}{\sqrt{2\pi}}e^{-x^2/2}$	Hermite $H_{e_k}(x)$	$H_{e_k}(x)/\sqrt{k!}$
Gamma	$x^a\, e^{-x}\, \mathbf{1}_{\mathbb{R}^+}(x)$	Laguerre $L_k^a(x)$	$L_k^a(x)/\sqrt{\frac{\Gamma(k+a+1)}{k!}}$
Beta	$\mathbf{1}_{]-1,1[}(x)\, \frac{(1-x)^a(1+x)^b}{B(a)\,B(b)}$	Jacobi $J_k^{a,b}(x)$	$J_k^{a,b}(x)/\mathfrak{J}_{a,b,k}$
		$\mathfrak{J}^2_{a,b,k} = \frac{2^{a+b+1}}{2k+a+b+1}\frac{\Gamma(k+a+1)\Gamma(k+b+1)}{\Gamma(k+a+b+1)\Gamma(k+1)}$	

1.3.2 Calculation of the coefficients

Different algorithms have been proposed in the literature for the calculation of the LRA coefficients in a non-intrusive manner (see *e.g.* Doostan et al. (2013); Mathelin (2014); Rai (2014); Validi (2014); Chevreuil et al. (2015)). A common attribute of these algorithms is the employment of an alternated least-squares (ALS) minimization scheme, which consists in sequentially solving a least-squares minimization problem along each dimension $1, \ldots, M$, while "freezing" the coefficients in all remaining dimensions. We herein employ the algorithm proposed by Chevreuil et al. (2015) and further investigated by Konakli and Sudret (2016b). This algorithm involves the progressive increase of the approximation rank by successively adding rank-one components up to a prescribed maximum.

Let us denote by $\widehat{\mathcal{M}}_r$ the rank-r approximation of $\mathcal{M}$. The corresponding model response is then approximated by:

$$\widehat{Y}_r = \widehat{\mathcal{M}}_r(\boldsymbol{X}) = \sum_{l=1}^{r} b_l w_l(\boldsymbol{X}), \tag{1.8}$$

where w_l represents the l-th rank-one component:

$$w_l(\boldsymbol{X}) = \prod_{i=1}^{M} \left(\sum_{k=0}^{p_i} z_{k,l}^{(i)} \, \phi_k^{(i)}(X_i) \right). \tag{1.9}$$

The employed algorithm involves a sequence of pairs of *correction* and *updating* steps, so that in the r-th correction step, the rank-one tensor w_r is built, whereas in the r-th updating step, the set of weighing factors $\{b_1, \ldots, b_r\}$ is determined. These steps are detailed below according to the presentation in Konakli and Sudret (2016b).

Correction step: To facilitate the mathematical description, we first introduce the following notation. Given a sample set $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathcal{D}_{\boldsymbol{X}}$ and a function a, we denote:

$$\| \, a \, \|_{\mathcal{X}} = \left(\frac{1}{n} \sum_{i=1}^{n} a^2(\boldsymbol{x}_i) \right)^{1/2}. \tag{1.10}$$

In the r-th correction step, the rank-one tensor w_r is obtained as the solution of the minimization problem:

$$w_r(\boldsymbol{X}) = \arg\min_{\omega \in \mathcal{W}} \left\| \mathcal{M} - \widehat{\mathcal{M}}_{r-1} - \omega \right\|_{\mathcal{X}}^2, \tag{1.11}$$

where $\mathcal{W}$ represents the space of rank-one tensors and the subscript $\mathcal{X}$ indicates that the minimization is carried over the experimental design. The sequence of $\widehat{\mathcal{M}}_r(\boldsymbol{X})$ is initiated by setting $\widehat{\mathcal{M}}_0(\boldsymbol{X}) = 0$. Eq. (1.11) is solved by means of an ALS scheme that involves sequential minimizations along each dimension $i = 1, \ldots, M$. In the minimization along the j-th dimension, the polynomial coefficients in all other dimensions are "frozen" at their current values and the coefficients $\boldsymbol{z}_r^{(j)} = \{z_{1,r}^{(j)} \ldots z_{p_j,r}^{(j)}\}$ are determined as:

$$\boldsymbol{z}_r^{(j)} = \arg\min_{\boldsymbol{\zeta} \in \mathbb{R}^{p_j+1}} \left\| \mathcal{M} - \widehat{\mathcal{M}}_{r-1} - \left(\prod_{i \neq j} v_r^{(i)} \right) \left(\sum_{k=0}^{p_j} \zeta_k \, \phi_k^{(j)} \right) \right\|_{\mathcal{X}}^2, \tag{1.12}$$

where:

$$v_r^{(i)}(X_i) = \sum_{k=0}^{p} z_{k,r}^{(i)} \, \phi_k^{(i)}(X_i). \tag{1.13}$$

To initiate the r-th correction step, one needs to assign arbitrary values to $v_r^{(i)}$, $i = 1, \ldots, M$; in the UQLAB implementation, the initial values $v_r^{(1)} = \ldots = v_r^{(M)} = 1$ are used. We emphasize that a correction step may involve several iterations over the set of dimensions $\{1, \ldots, M\}$. Konakli and Sudret (2016b) showed that the number of iterations performed may be critical for the LRA accuracy and proposed a stoppping criterion that combines the number of iterations I_r with the decrease $\Delta\widehat{err}_r$ of an error measure in two successive iterations. The employed error measure is the relative empirical error, given by:

$$\widehat{err}_r = \frac{\left\| \mathcal{M} - \widehat{\mathcal{M}}_{r-1} - w_r \right\|_{\mathcal{X}}^2}{\widehat{\mathrm{Var}}\left[\mathcal{Y}\right]}. \tag{1.14}$$

In the above equation, $\widehat{\mathrm{Var}}[\mathcal{Y}]$ represents the empirical variance of the set consisting of the model responses at the experimental design. Accordingly, the algorithm exits the r-th correction step if either I_r reaches the maximum allowed value $I_{\max}$, or $\Delta\widehat{err}_r$ becomes smaller than a prescribed threshold $\Delta\widehat{err}_{\min}$. This stopping criterion with appropriate values for $I_{\max}$ and $\Delta\widehat{err}_{\min}$ is used in the UQLAB implementation.

Updating step: After the completion of a correction step, the algorithm moves to an updating step, in which the set of coefficients $\boldsymbol{b} = \{b_1, \ldots, b_r\}$ is obtained as the solution of the minimization problem:

$$\boldsymbol{b} = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^r} \left\| \mathcal{M} - \sum_{l=1}^{r} \beta_l w_l \right\|_{\mathcal{X}}^2 . \tag{1.15}$$

Note that in each updating step the size of vector $\boldsymbol{b}$ is increased by one. In the r-th updating step, the value of the new element b_r is determined for the first time, whereas the values of the existing elements $\{b_1, \ldots, b_{r-1}\}$ are updated.

Construction of a rank-R decomposition in the form of Eq. (1.4) requires repeating pairs of a correction and an updating step for $r = 1, \ldots, R$. Therefore, the full construction relies on the solution of several small-size minimization problems; in particular, one needs to solve M minimization problems of size $\{p_i + 1,\ i = 1, \ldots, M\}$ in each iteration of a correction step (note that $p < 20$ in typical applications) and one minimization problem of size r in the r-th updating step (recall that small ranks are of interest in LRA). We herein emphasize the distinct difference to the construction of PCE, which relies on a single large-size minimization problem (see UQLAB User Manual – Polynomial Chaos Expansions for further details).

Because of the small number of unknowns involved in Eq. (1.12) and Eq. (1.15), these minimization problems can be efficiently solved with the ordinary least-squares (OLS) technique. An alternative approach employed by Chevreuil et al. (2015) is to substitute these equations with the respective regularized problems. For instance, the L_1-regularized problems read:

$$\boldsymbol{z}_r^{(j)} = \arg\min_{\boldsymbol{\zeta}\in\mathbb{R}^{p_j+1}} \left\| \mathcal{M} - \widehat{\mathcal{M}}_{r-1} - \left(\prod_{i\neq j} v_r^{(i)} \right) \left(\sum_{k=0}^{p_j} \zeta_k\, \phi_k^{(j)} \right) \right\|_{\mathcal{X}}^2 + \lambda \|\boldsymbol{\zeta}\|_1 \tag{1.16}$$

and

$$\boldsymbol{b} = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^r} \left\| \mathcal{M} - \sum_{l=1}^{r} \beta_l w_l \right\|_{\mathcal{X}}^2 + \lambda \|\boldsymbol{\beta}\|_1 . \tag{1.17}$$

UQLAB provides the possibility of solving Eq. (1.16) and Eq. (1.17) with the least-angle regression (LARS) method (see UQLAB User Manual – Polynomial Chaos Expansions for information on the LARS algorithm).

1.3.3 Selection of optimal rank

The progressive construction of LRA described in Section 1.3.2 yields a set of metamodels of increasing rank. Executing the algorithm for $R = r_{\max}$ will result in a set of LRA with ranks $r = 1, \ldots, r_{\max}$; one may then use error-based criteria to select the metamodel with

the optimal rank $R^{\text{opt}} \in \{1, \ldots, r_{\max}\}$. Alternatively, the progressive increase of the rank may be stopped earlier at $r_{\text{stop}} < r_{\max}$, if there is sufficient evidence that further addition of rank-one components does not improve the metamodel accuracy. This option of "early stop" is available in UQLAB and is based on monitoring the metamodel error after each addition of a rank-one term. If the error consistently increases for a predefined number of additions, then the algorithm is terminated. Further details on the UQLAB usage are given in the next chapter of the manual.

The error measure used in UQLAB for the selection of the optimal rank is obtained via n-fold cross validation. The procedure first requires partitioning the experimental design in n sets of approximately equal size. LRA of progressively increasing rank are built considering all but one of the partitions, which comprise the training set $\mathcal{TR}$. The excluded or testing set $\mathcal{TS}$, is used to evaluate the error of the LRA built with the training set:

$$\widehat{err} = \frac{\left\| \mathcal{M} - \widehat{\mathcal{M}}^{\text{LRA},\mathcal{TR}} \right\|_{\mathcal{TS}}^2}{\widehat{\text{Var}}\left[\mathcal{Y}_{\mathcal{TS}}\right]}, \tag{1.18}$$

where $\widehat{\mathcal{M}}^{\text{LRA},\mathcal{TR}}$ denotes the LRA metamodel built with the training set and $\widehat{\text{Var}}\left[\mathcal{Y}_{\mathcal{TS}}\right]$ denotes the empirical variance of the set consisting of the model evaluations at the testing set. By alternating through the n sets, n metamodels are obtained in this way; their average error provides an estimate of the generalization error of the metamodel, which is used to control the early-stop option and select the optimal rank R^{opt}. Once the optimal rank has been identified, a new decomposition of rank R^{opt} is built using the full experimental design. Numerical investigations by Konakli and Sudret (2016b) have shown that the 3-fold cross validation proposed by Chevreuil et al. (2015) is sufficiently accurate for model selection purposes in the context of canonical LRA.

A different error estimate relying on the experimental design is the relative empirical error:

$$\widehat{err}_E = \frac{\left\| \mathcal{M} - \widehat{\mathcal{M}}^{\text{LRA}} \right\|_{\mathcal{E}}^2}{\widehat{\text{Var}}\left[\mathcal{Y}\right]}, \tag{1.19}$$

i.e. the empirical error normalized with the variance of the set of model responses at the experimental design. The empirical error is based on a simpler computation than cross-validation, but it can severely overestimate the actual error in cases of overfitting and is thus avoided. Although it is not used for model selection, the relative empirical error of the final LRA is provided by UQLAB as a metamodel property.

1.3.4 Selection of optimal polynomial degree

In the current implementation of canonical LRA in UQLAB, a common polynomial degree is considered in all dimensions, *i.e.* $p_1 = \ldots = p_M = p$ in Eq. (1.4). The error measure based on n-fold cross-validation, described in the previous paragraph, is also used to determine the optimal value of the common degree p among a candidate set. Similarly to the rank selection,

a progressive increase of the polynomial degree with an early-stop option may be employed. Further details on the selection of the optimal polynomial degree in UQLAB are given in the next chapter of the manual.

1.3.5 *A posteriori* error estimation

After the LRA metamodel is set up, its predictive accuracy on new data can be assessed by using the so-called *validation error*. It is calculated as the relative generalization error on an independent set of inputs and outputs $[\boldsymbol{\mathcal{X}}_{Val},\ \mathcal{Y}_{Val} = \mathcal{M}(\boldsymbol{\mathcal{X}}_{Val})]$:

$$\epsilon_{Val} = \frac{N-1}{N} \left[\frac{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}_{Val}^{(i)}) - \mathcal{M}^{\mathrm{LRA}}(\boldsymbol{x}_{Val}^{(i)}) \right)^2}{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}_{Val}^{(i)}) - \hat{\mu}_{Y_{Val}} \right)^2} \right] \tag{1.20}$$

where $\hat{\mu}_{Y_{Val}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{M}(\boldsymbol{x}_{Val}^{(i)})$ is the sample mean of the validation set response. This error measure is useful to compare the performance of different surrogate models when evaluated on the same validation set.

1.4 Moments of an LRA

Due to the orthonormality of the univariate polynomials that form the LRA basis (see Eq. (1.7)), the mean and variance of the metamodel can be obtained analytically in terms of the polynomial coefficients and the weighing factors. In particular, the mean and variance of the LRA response are respectively given by (Konakli and Sudret, 2016a):

$$\mu^{\mathrm{LRA}} = \mathbb{E}\left[\mathcal{M}^{\mathrm{LRA}}(\boldsymbol{X})\right] = \sum_{l=1}^{R} b_l \left(\prod_{i=1}^{M} z_{0,l}^{(i)} \right) \tag{1.21}$$

and

$$\left(\sigma^{\mathrm{LRA}}\right)^2 = \sum_{l=1}^{R} \sum_{l'=1}^{R} b_l\, b_{l'} \left(\left(\prod_{i=1}^{M} \left(\sum_{k=0}^{p_i} z_{k,l}^{(i)}\, z_{k,l'}^{(i)} \right) \right) - \left(\prod_{i=1}^{M} z_{0,l}^{(i)}\, z_{0,l'}^{(i)} \right) \right). \tag{1.22}$$

1.5 Other post-processing techniques

Once the LRA metamodel is developed, performing evaluations on new samples of the input random vector is essentially costless from a computational viewpoint. This property allows one to efficiently conduct any type of statistical analysis of the model response, *e.g.* PDF estimation, evaluation of statistical moments and confidence intervals, computation of exceedance probabilities and so forth. Another important property is that the analysis-of-variance (ANOVA) decomposition of the model response can be obtained analytically in terms of the LRA coefficients (Konakli and Sudret, 2016a). Thus, once the LRA representation of the model response is available, global sensitivity analysis can be performed at nearly zero

additional computational cost. For information on computing the LRA-based global sensitivity indices with UQLAB, the interested reader is referred to the UQLAB User Manual – Sensitivity Analysis module.

Chapter 2

Usage

In this chapter, a reference problem is employed to demonstrate step-by-step the computation of canonical LRA with UQLAB and explain the relevant options.

2.1 Reference problem: Beam deflection

We consider the simply-supported beam shown in Figure 1. The beam has a constant rectangular cross-section of width b and height h, a span of length L and Young modulus E; it is subjected to a uniform vertical load of magnitude p. The response quantity of interest is the mid-span deflection given by:

$$V = \frac{5}{32}\frac{pL^4}{Ebh^3}, \tag{2.1}$$

The aforementioned input parameters of the model are represented by independent random variables with distributions, mean and coefficient of variation (CoV) values listed in Table 2. The random input of the problem is thus described by the random vector $\boldsymbol{X} = \{b, h, L, E, P\}$ of dimension $M = 5$.

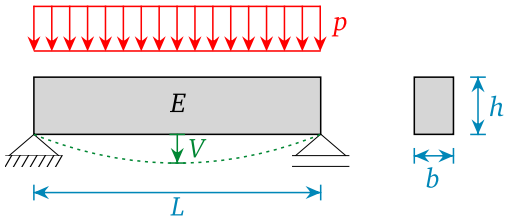

Figure 1: Beam structure.

Table 2: Beam deflection: Distributions of input random variables.

Variable	Distribution	mean	CoV
b (m)	Lognormal	0.15	0.05
h (m)	Lognormal	0.3	0.05
L (m)	Lognormal	5	0.01
E (MPa)	Lognormal	$30,000$	0.15
P (MN/m)	Lognormal	0.01	0.20

2.2 Problem set-up

A canonical LRA with polynomial basis and common polynomial degree p in all dimensions reads (see Eq (1.4)):

$$\widehat{\mathcal{M}}_R(\boldsymbol{X}) = \sum_{l=1}^{R} b_l \left(\prod_{i=1}^{M} \left(\sum_{k=0}^{p} z_{k,l}^{(i)} \, \phi_k^{(i)}(x_i) \right) \right), \tag{2.2}$$

The main ingredients of an analysis with a metamodel of this class are:

- A model to surrogate $Y = \mathcal{M}(\mathbf{X})$;
- A probabilistic input model (random input vector $\mathbf{X}$);
- The polynomial types in the different dimensions and the common maximum polynomial degree p;
- The rank R of the approximation;
- The polynomial coefficients $z_{k,l}^{(i)}$ and weighing factors b_l.

The first two items, which are common in any metamodel analysis, must be defined by the user, as described in Section 2.2.1 and Section 2.2.2 that follow. The last three items represent elements of the LRA metamodel that are determined by UQLAB as will be explained later in this chapter.

2.2.1 Full model

In the current example, the computational model, given in Eq. (2.1), is implemented as a MATLAB `m-file` in:

`Examples/SimpleTestFunctions/uq_SimplySupportedBeam.m`

This is added to UQLAB with the following commands:

```
ModelOpts.mFile = 'uq_SimplySupportedBeam';
myModel = uq_createModel(ModelOpts);
```

For more details about the configuration options available for a model, please refer to the UQLAB User Manual – the MODEL module.

2.2.2 Probabilistic input model

The probabilistic input model given in Table 2 is created as follows:

```
Input.Marginals(1).Name = 'b';
Input.Marginals(1).Type = 'Lognormal';
Input.Marginals(1).Moments = [0.15 0.0075];

Input.Marginals(2).Name = 'h';
Input.Marginals(2).Type = 'Lognormal';
Input.Marginals(2).Moments = [0.3 0.015];
```

```
Input.Marginals(3).Name = 'L';
Input.Marginals(3).Type = 'Lognormal';
Input.Marginals(3).Moments = [5 0.05];

Input.Marginals(4).Name = 'E';
Input.Marginals(4).Type = 'Lognormal';
Input.Marginals(4).Moments = [3e4 4500];

Input.Marginals(5).Name = 'P';
Input.Marginals(5).Type = 'Lognormal';
Input.Marginals(5).Moments = [0.01 0.002];

myInput = uq_createInput(Input);
```

For more details about the configuration options available for an INPUT object, please refer to the UQLAB User Manual – the INPUT module.

2.3 LRA analysis using default options

2.3.1 Building the LRA

The LRA module creates a MODEL object that can be used as any other model. The basic options common to any LRA metamodel are:

```
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'LRA';
```

The only elements of the LRA metamodel that must be defined in the UQLAB framework are the user's preferences on the allowable degrees and ranks, as described in the next paragraphs of this section. Note that the polynomial types will be automatically determined by UQLAB according to the probabilistic input model. UQLAB will use default options to calculate the LRA coefficients for the defined range of degrees and ranks and will eventually return the metamodel characterized by the lowest error estimate.

The polynomial degree p may be specified as a single value or a range of possible values in the `Degree` field of the `MetaOpts` configuration variable. In the latter case, the optimal degree will be selected according to an appropriate error measure (see Section 2.5.2.1). For instance, to indicate that p can take values within the range $\{2, \ldots, 10\}$, one sets:

```
MetaOpts.Degree = 2:10;
```

In this case, the underlying routine will build LRA with sequentially increasing polynomial degrees, starting from the lowest possible value $p = 2$. In order to optimize the computation time, the search for the optimal polynomial degree will, by default, terminate if the specified error measure increases for two consecutive polynomial degrees. It is possible to configure UQLAB to search all specified degrees or stop only if the error measure increases for a selected number of degrees (Section 2.5.2.2).

The rank of the metamodel is defined in the `Rank` field of the `MetaOpts` configuration vari-

able. In order to specify that UQLAB should test ranks 1 to 20 one should specify:

```
MetaOpts.Rank = 1:20;
```

In this case, the optimal rank $R^{\text{opt}} \in \{1, \ldots, r_{\max}\}$ will be identified according to an appropriate error measure. The algorithm will eventually return the LRA metamodel with the identified optimal rank R^{opt}. Similarly to the case of the polynomial degree, the search for the optimal rank will, by default, terminate if the error measure increases for two consecutive ranks. It is possible to configure UQLAB to search all specified ranks or stop only if the error measure increases for a selected number of ranks (Section 2.5.2.2). In order to build a metamodel of a certain rank R (without performing a search for the optimal), one may simply set the desired rank (single value) in the `MetaOpts.Rank` field.
After the options concerning the LRA formulation have been specified, the only remaining element to be defined is the experimental design. We herein consider an experimental design of size $N = 100$ sampled with latin hypercube sampling (LHS), which is specified in UQLAB as follows:

```
MetaOpts.ExpDesign.NSamples = 100;
MetaOpts.ExpDesign.Sampling = 'LHS';
```

Finally, the metamodel is created by simply invoking:

```
myLRA = uq_createModel(MetaOpts);
```

2.3.2 Accessing basic information on the LRA

Once the model is created, a report with basic information on the LRA can be printed out with the command:

```
uq_print(myLRA);
```

which, in our example, produces the following output:

```
%------------ Canonical LRA output ------------%
   Number of input variables:          5
   Full model evaluations:           100
   Degree of polynomials used:         4
   Rank:                               3
   Correction Step:
       Regression Method:            OLS
   Updating Step:
       Regression Method:            OLS
   CVError:                    1.3045e-05
   Mean value:                    0.0084
   Standard deviation:            0.0025
   Coef. of variation:            30.334%
%----------------------------------------------%
```

As can be seen in the above report, the algorithm has identified the values $p = 4$ and $R = 3$ as the optimal polynomial degree and rank, respectively. Note that these values correspond to a single random partition of the experimental design in testing and training sets in 3-fold cross-validation (the default method for error estimation). In other words, different combinations may be identified in repeated runs of the algorithm (see Konakli and Sudret (2016b) for the variability in rank selection due to the random partitioning of the experimental design). The report also includes the 3-fold CV error (the default error estimate) of the metamodel as well as the mean, standard deviation and CoV of the response. Additional information listed is the size of the input vector and the experimental design and the regression method used in the correction and the updating steps.

The `uq_display` command allows the user to visualize the LRA estimation of the experimental design against the original model responses. In our example, using the command

```
uq_display(myLRA);
```

produces the image in Figure 2.

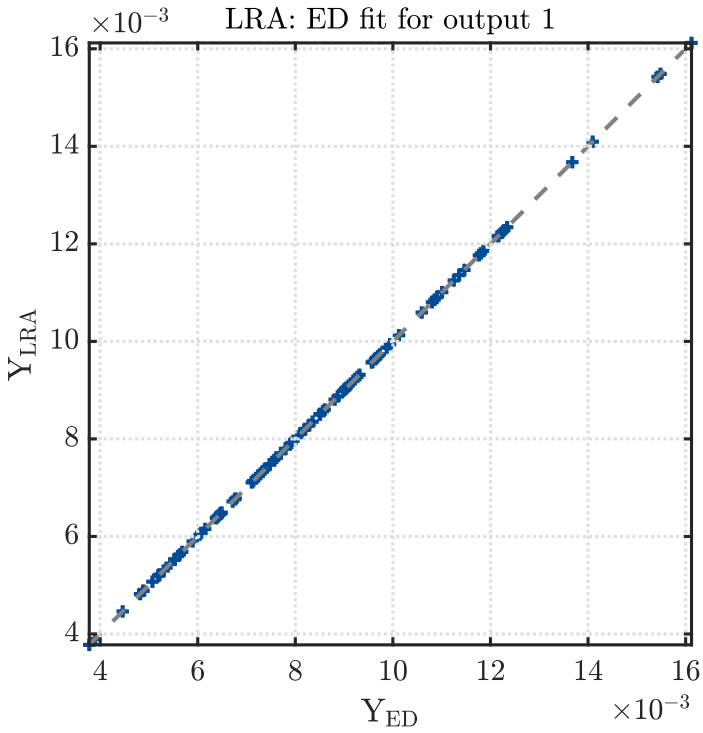

Figure 2: LRA display: Original *vs.* estimated experimental design responses

2.4 Accessing the detailed results

2.4.1 Basis, coefficients and moments

All the information needed to evaluate Eq. (1.4) is available in the output structure `myLRA.LRA`:

```
myLRA.LRA
ans =
```

```
         Basis: [1x1 struct]
  Coefficients: [1x1 struct]
       Moments: [1x1 struct]
```

The structure `myLRA.LRA.Basis` contains information on the polynomial types, the polynomial degree and the metamodel rank. For our example problem, this structure includes the following elements:

```
myLRA.LRA.Basis
ans =
        PolyTypes: {5x1 cell}
  PolyTypesParams: {5x1 cell}
      PolyTypesAB: {{1x2 cell}  {1x2 cell}  {1x2 cell}  {1x2 cell}
                    {1x2 cell}}
           Degree: 4
            Rank: 2
```

The first three cell arrays refer to the polynomial types: the arrays `Basis.PolyTypes`, `Basis.PolyTypesParams` and `Basis.PolyTypesAB` respectively contain the names of the M polynomial families, the parameters used in the computation of the polynomials and the corresponding recurrence coefficients. The `Basis.Degree` and `Basis.Ranks` elements indicate the polynomial degree and rank of the final metamodel; as noted in Section 2.3.2, these values correspond to a single random partition of the experimental design in testing and training sets.

The values of the polynomial coefficients $\{z_{k,r}^{(i)},\ i = 1, \dots, M,\ k = 0, \dots, p,\ r = 1, \dots, R\}$ and the weighing factors $\{b_r,\ r = 1, \dots, R\}$ are contained in the structure `myLRA.LRA.Coefficients`:

```
myLRA.LRA.Coefficients
ans =
  z: {[5x5 double]  [5x5 double]}
  b: [2x1 double]
```

The cell array `Coefficients.z` contains R cells, with the r-th cell storing the coefficients $\{z_{k,r}^{(i)},\ i = 1, \dots, M,\ k = 0, \dots, p\}$ of the r-th rank-one component. The polynomial coefficients of a rank-one component are stored in a matrix with $p + 1$ rows, corresponding to the polynomial degrees $\{0, \dots, p\}$, and M columns corresponding to the dimensions $\{1, \dots, M\}$. For example, the coefficients $\{z_{k,1}^{(2)},\ k = 0, \dots, p\}$ of the 1st rank-one component along the 2nd dimension (herein corresponding to variable h) can be accessed with the command:

```
myLRA.LRA.Coefficients.z{1}(:,2)
```

It was shown in Section 1.4 that the LRA mean and variance can be computed from the metamodel coefficients only. The corresponding values are contained in the structure `myLRA.LRA.Moments`:

```
myLRA.LRA.Moments
ans =
  Mean: 0.0084
   Var: 6.4448e-06
```

2.4.2 Error measures

The structure `Error` includes estimates of the metamodel error. In particular, it contains the values of (i) the specified error measure in the configuration variable `MetaOpts.GenError`, which is by default the 3-fold CV error (unless specified otherwise), and (ii) the normalized empirical error. These error measures for a single run of the present example are listed below:

```
myLRA.Error
ans =
  SelectedCVScore: 2.7426e-07
     normEmpError: 9.5273e-10
```

As explained earlier, different runs may yield different error estimates due to the randomness in the partition of the experimental design in testing and training tests.

2.4.3 Experimental design

The `ExpDesign` structure contains information on the experimental design used to build the LRA metamodel. In particular, it includes the number of samples `NSamples`, the sampling strategy `Sampling`, the experimental design `X`, the corresponding input samples `U` in the considered standard space (which depends on the chosen families of orthogonal polynomials) and the set of model responses `Y` at the experimental design. The content of this structure for the current example is displayed below:

```
myLRA.ExpDesign
ans =
  NSamples: 100
  Sampling: 'LHS'
  ED_Input: [1x1 uq_input]
         X: [100x5 double]
         U: [100x5 double]
         Y: [100x1 double]
```

2.4.4 Additional information

The structure `Internal` contains additional information related to the computation of the LRA. The content of this structure for the current example is displayed below:

```
myLRA.Internal
ans =
             Runtime: [1x1 struct]
             Display: 1
               Input: [1x1 uq_input]
           FullModel: [1x1 uq_model]
  ComputationalOpts: [1x1 struct]
            eventLog: [1x3 struct]
           ExpDesign: [1x1 struct]
            StepData: [1x1 struct]
              Scores: [1x1 struct]
```

The structure `Internal.StepData` includes information for all the correction and updating steps performed in the construction of the final LRA metamodel, which can be useful for a more in-depth investigation of the process. In particular, it contains:

- the `CorrStep` structure, which includes (i) the normalized empirical error `errE_iter` and (ii) the differential error `Derr` after each iteration of a correction step, and

- the `UpdateStep` structure, which includes the mean residual over the points of the experimental design after each updating step.

For instance, the command:

```
myLRA.Internal.StepData.CorrStep(1).Iterations(2)
```

will provide the errors `errE_iter` and `Derr` after the second iteration of the first correction step, while the command:

```
myLRA.Internal.StepData.UpdateStep(2)
```

will provide the mean residual after the second updating step.
By accessing the `myLRA.Internal.Scores` structure, the user can inspect the cross-validation errors (scores) for each combination of rank and degree tested. As an example, we demonstrate below the content of this structure for the same beam problem and the default options, but for the simple illustrative case when the rank and degree are allowed to vary in the ranges $\{1,2,3,4,5\}$ and $\{2,3\}$, respectively:

```
myLRA.Internal.Scores
ans =
      R: [1 2 3 4 5 1 2 3]
      p: [2 2 2 2 2 3 3 3]
  Score: [1.7160e-04 0.0039 0.0034 0.0048 0.0019 1.1472e-06
  1.5982e-06 1.0319e-05]
```

In the above results, the absence of certain combinations of rank and degree belonging in the allowable ranges is due to satisfaction of the early-stop criterion.

2.5 Advanced options

The LRA module provides additional advanced configuration options that the user might find useful in certain contexts. These are described in the following subsections.

2.5.1 Polynomial types

For constructing the LRA basis, UQLAB will, by default, use appropriate polynomials according to the distributions of the input variables. These are either standard Wiener-Askey polynomials (Legendre, Hermite, Laguerre, Jacobi) or polynomials orthogonal to arbitrary marginals based on the Stieltjes procedure (Gautschi, 2004). The default univariate polynomial are given in Table 3.
In our reference beam-deflection problem, all five input variables follow lognormal distributions; thus, in the example case of Section 2.3, where the default options are considered, the univariate functions in all dimensions are made of Hermite polynomials.
Alternatively, the user may enforce the type of univariate polynomials by specifying the `PolyTypes` field in the `MetaOpts` configuration variable. For example, in order to instruct UQLAB to compute arbitrary polynomials that are orthogonal to the input distributions in all dimensions, the user needs to specify:

Table 3: Default univariate polynomial types used in UQLAB w.r.t. input distributions.

Input PDF $f_{X_i}(X_i)$	Univariate polynomial family
$\mathcal{U}(a,b)$	Legendre
$\mathcal{N}(\mu,\sigma)$	Hermite
$\mathcal{LN}(\mu,\sigma)$	Hermite
$\Gamma(\lambda,\kappa)$	Laguerre(λ,κ)
$\mathcal{B}(r,s,a,b)$	Jacobi(r,s)
Other	Stieltjes

```
MetaOpts.PolyTypes = {'Arbitrary', 'Arbitrary', 'Arbitrary',...
                      'Arbitrary', 'Arbitrary'};
```

In a case when the use of Laguerre or Jacobi polynomials would be appropriate, the user should also specify the parameters of the polynomials. For instance, for a hypothetical model with $M = 3$, in order to use Jacobi and Laguerre polynomials in the first two dimensions and Hermite polynomials in the third dimension, one may specify:

```
MetaOpts.PolyTypes       = {'Jacobi', 'Laguerre', 'Hermite'};
MetaOpts.PolyTypesParams = { [2 3],    [3 4],      []       };
```

Note that the dimension of the `MetaOpts.PolyTypes` cell-array must agree with that of the input model.

2.5.2 Computational options

2.5.2.1 Error estimation

The method of estimation of the LRA generalization error is defined in the configuration field `MetaOpts.GenError`. This error estimate provides the criterion for the selection of the optimal combination of rank and polynomial degree. At the present version of the module, the only available method for error estimation is n-fold cross validation, corresponding to the default `'CV'` value of the field `GenError.Method`. The user has the option to select the number of the experimental design partitions n at the `Parameters.NFolds` field. For instance, error estimation via 5-fold cross-validation is specified with the commands:

```
MetaOpts.GenError.Method = 'CV';
MetaOpts.GenError.Parameters.NFolds = 5;
```

The default number of partitions is 3, unless explicitly specified as above.

2.5.2.2 Rank and degree selection

There are three adaptive strategies currently implemented in the LRA module for the selection of rank and maximum polynomial degree. These strategies are selected by assigning the appropriate value to the field `MetaOpts.Adaptivity`, as explained below:

- **Rank adaptivity:** It is selected with the `'all_d_adapt_r'` option. In this case, the candidate combinations of rank and polynomial degree are obtained by combining increasing polynomial degrees with the range of allowable ranks. This is the default option, used when the user does not specify any strategy.
- **Degree adaptivity:** It is selected with the `'all_r_adapt_d'` option. In this case, the candidate combinations of rank and polynomial degree are obtained by combining increasing ranks with the range of allowable polynomial degrees.
- **Rank and degree adaptivity:** It is selected with the `'adapt_r_d'` option. In this case, the rank and/or degree are only increased if their increase reduces the error estimate.

For instance, the user can specify the degree-adaptive strategy, simply by setting:

```
MetaOpts.Adaptivity = 'all_r_adapt_d';
```

The adaptation strategies use the options of the `.DegSelection` and `.RankSelection` fields, which are described below.

Rank selection options: The user can deactivate the default early-stop option for the rank search by setting the field `EarlyStop` of the `MetaOpts.RankSelection` configuration variable to 0. In order to retain the early-stop option, but modify the number of steps that will lead to termination of the search, the user can simply specify the `EarlyStopSteps` field of the `MetaOpts.RankSelection` configuration variable. The default values for the aforementioned fields are:

```
MetaOpts.RankSelection.EarlyStop = 1;
MetaOpts.RankSelection.EarlyStopSteps = 2;
```

Degree selection options: In order to perform the search over the entire specified range of polynomial degrees, regardless of the error estimate, one needs to set the field `EarlyStop` of the `MetaOpts.DegSelection` configuration variable to 0. Alternatively, one may retain the early-stop option, but control the number of steps with increasing error that will lead to termination of the search; the latter can be specified at the `EarlyStopSteps` field of the `MetaOpts.DegSelection` configuration variable. The default values for the aforementioned fields are:

```
MetaOpts.DegSelection.EarlyStop = 1;
MetaOpts.DegSelection.EarlyStopSteps = 2;
```

2.5.2.3 Calculation of the coefficients

As explained in Section 1.3.2, the set of polynomial coefficients $\{z_{k,r}^{(i)},\ i = 1, \ldots, M,\ k = 0, \ldots, p\}$ for the r-th rank-one component is evaluated in the r-th correction step, whereas the set of weighing factors $\{b_l,\ l = 1, \ldots, r\}$ is evaluated/updated in the r-th updating step.

The specification of the relevant options for the two types of steps performed successively in the algorithm are described next.

The options pertinent to the correction step are specified as fields of the configuration variable `Metaopts.CorrStep`. In particular, these options include the values of the stopping parameters $I_{\max}$ and $\Delta\widehat{err}_{\min}$ and the minimization method used to solve Eq. (1.12). Corresponding values of these parameters are specified in the fields `MinDerrStop`, `MaxIterStop` and `Method`, as shown in the following example set-up:

```
MetaOpts.CorrStep.MaxIterStop = 50;
MetaOpts.CorrStep.MinDerrStop = 10^-6;
MetaOpts.CorrStep.Method = 'OLS';
```

The default values used if the above are not specified by the user are $I_{\max} = 100$ and $\Delta\widehat{err}_{\min} = 10^{-6}$ for the stopping criterion and `'OLS'` for the minimization method.

The updating step involves only the option of the minimization method for solving Eq. (1.15). This is defined in the field `Method` of the configuration variable `MetaOpts.UpdateStep`:

```
MetaOpts.UpdateStep.Method = 'OLS';
```

The default value is `'OLS'`.

2.5.3 Experimental design

- **Specify a sampling strategy:** The default sampling method for the experimental design is latin hypercube sampling. A different sampling strategy may be defined through the `ExpDesign.Sampling` option. For instance, the following specifies sampling from a Sobol' pseudo-random sequence:

  ```
  MetaOpts.ExpDesign.Sampling = 'Sobol';
  ```

- **Manually specify an experimental design:** In many practical situations, a user needs to develop a metamodel from given data. Similarly to the case of PCE, there are two ways to import data in a UQLAB LRA MODEL:

  - Specify the values of `ExpDesign.X` and `ExpDesign.Y` directly. Assuming such values are stored in the local variables `X_ED` and `Y_ED`, they can be imported in UQLAB as follows:

    ```
    MetaOpts.ExpDesign.X = X_ED;
    MetaOpts.ExpDesign.Y = Y_ED;
    ```

  - Specify a data file, *e.g.* `'mydata.mat'`:

    ```
    MetaOpts.ExpDesign.DataFile = 'mydata.mat';
    ```

    Currently, only mat-files containing two variables `X` and `Y` can be automatically loaded in UQLAB.

Important note: When an experimental design is specified manually there is no need to create a MODEL object as in Section 2.2.1. However, as in a PCE analysis, an INPUT module with an input random vector compatible with the provided experimental design *must* be defined. This is due to the fact that the probability distributions of the input variables are needed to calculate the LRA coefficients.

A comprehensive list of the options available for the calculation of the experimental design of LRA can be found in Table 11.

2.5.4 Use of a validation set

If a validation set is provided (see Table 12 in Section 3.1.6.), UQLAB automatically computes the validation error given in Eq. (1.20). To provide a validation set, the following command shall be used:

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

The value of the validation error is stored in `myLRA.Error.Val` next to the other error measures (see Table 20 in Section 3.2.3) and will also be displayed when typing `uq_print(myLRA)`.

2.6 Manually specify inputs and computational models

The UQLAB framework allows one to create many INPUT and MODEL objects in the same session (see UQLAB User Manual – the MODEL module and UQLAB User Manual – the INPUT module). In this respect, the current module follows a similar pattern with the PCE module and thus, the presentation herein is according to the respective section in the UQLAB User Manual – Polynomial Chaos Expansions.
The default behaviour of the LRA module is to use as probabilistic input (resp. computational model) the last created INPUT (resp. MODEL) object. This behaviour can be altered by manually specifying the desired objects in the configuration as follows:

- **Specify an INPUT object:** an INPUT object `myInput` can be specified with:

```
MetaOpts.Input = myInput;
```

- **Specify a MODEL object:** a MODEL object `myModel` can be specified with:

```
MetaOpts.FullModel = myModel;
```

2.7 LRA of vector-valued models

The example presented so far in this chapter dealt with a scalar-valued model. For vector-valued models, the current module follows a similar pattern with the PCE module and thus,

the presentation herein is according to the respective section in the UQLAB User Manual – Polynomial Chaos Expansions.

When the model (or the experimental design, if manually specified) has multi-component outputs (denoted by N_{out}), UQLAB develops an independent LRA for each output component based on the shared experimental design. No additional configuration is needed to enable this behaviour. A UQLAB example on LRA with multi-component outputs can be found in:

`Examples/LRA/uq_Example_LRA_04_MultipleOutputs.m`

Running a LRA calculation on a multi-component output model will result in a multi-component output structure. As an example, a model with 9 outputs will produce the following output structure:

```
myLRA.LRA
ans =
  1x9 struct array with fields:
      Basis
      Coefficients
      Moments
```

Each element of the `LRA` structure is functionally identical to its scalar counterpart in Section 2.4.

Similarly, the `myLRA.Error` structure becomes a multi-element structure:

```
myLRA.Error
ans =
  1x9 struct array with fields:
      SelectedCVScore
      normEmpError
```

2.8 Using an LRA as a predictor

After a LRA MODEL object is created in UQLAB, it can be used just like an ordinary model (for details, see the UQLAB User Manual – the MODEL module). For instance, it can be used to predict the response(s) of interest at new points outside of the experimental design. In order to assess the accuracy of the predictions in our example, let us compare the responses `YLRA` of the LRA with the responses `Y` of the actual model on an input sample set of size $N = 10^5$. The respective commands in UQLAB are listed below (for details on how to use the input module for sampling, see the UQLAB User Manual – the INPUT module):

```
N = 10^5;
X = uq_getSample(N);
Y = uq_evalModel(myModel,X);
YLRA = uq_evalModel(myLRA,X);
```

The histogram and scatter plots of the `Y` and `YLRA` vectors are given in Figure 3. The LRA responses practically coincide with the responses of the actual model, indicating the high accuracy of the herein developed metamodel.

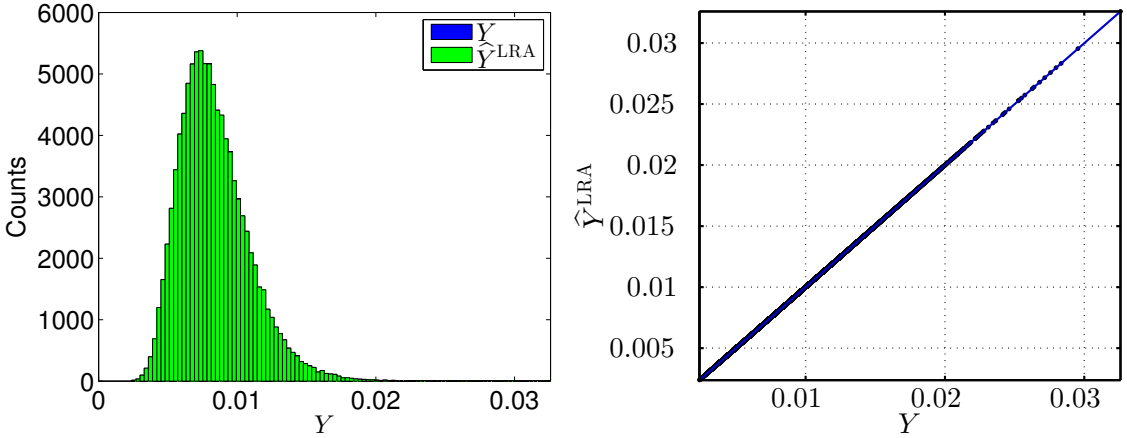

Figure 3: Histogram and scatter plots of true *vs.* metamodelled beam-deflection responses for an input sample of size $N = 10^5$.

2.9 Manually specifying LRA parameters (*predictor-only* mode)

It is also possible to use the LRA module in UQLAB to build custom LRA-based models that can be used as MODEL objects as in Section 2.8. This allows *e.g.* to import a metamodel calculated with another software within the UQLAB framework, or even or to create one *ad-hoc*. With regard to this feature, the current module follows a pattern similar to the PCE module and thus, the presentation herein is according to the respective section in the UQLAB User Manual – Polynomial Chaos Expansions.

In the following, we exemplify how to create a custom LRA with the following characteristics:

- standard normal input variables;
- rank-2 approximation metamodel;
- polynomial basis comprising up to second-degree Hermite polynomials;
- a set of random polynomial coefficients and weighing factors.

```
% Startup the framework
uqlab

% Create an Input object
for ii = 1:2
  inputOpts.Marginals(ii).Type = 'Gaussian';
  inputOpts.Marginals(ii).Moments = [0 1];
end
myInput = uq_createInput(inputOpts);

% Create a custom LRA
predopts.Type = 'Metamodel';
predopts.MetaType = 'LRA';
predopts.Method = 'Custom';
predopts.Input = myInput;

% Specify the LRA basis
predopts.LRA.Basis.PolyTypes = {'Hermite','Hermite'};
```

```
predopts.LRA.Basis.Rank = 2;
predopts.LRA.Basis.Degree = 2;

% Generate a set of random polynomial coefficients
% and weighing factors
predopts.LRA.Coefficients.b =  rand(2,1);
predopts.LRA.Coefficients.z = {rand(3,2),rand(3,2)};

% Create the metamodel
myLRA = uq_createModel(predopts);

% Evaluate the model on a sample of the input
X = uq_getSample(1000);
Y = uq_evalModel(X);
```

Note that UQLAB takes care automatically of any isoprobabilistic transformation between the probabilistic input model and the space onto which the specified polynomial families are orthogonal.
When the desired metamodel has more than one output, it is sufficient to specify the same information for each of the outputs by adding an index `i` to the `MetaOpts.LRA(i)` structure.

2.10 Using LRA with 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 LRA 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.
To set a parameter to constant, the following command can be used (See UQLAB User Manual – the INPUT module):

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

Furthermore, when the standard deviation of a parameter is set to zero, UQLAB automatically sets this parameter's marginal to the type `Constant`. For example, the following uniformly distributed variable whose upper and lower bounds are identical is automatically set to a constant with value 1:

```
inputOpts.Marginals.Type = 'Uniform' ;
inputOpts.Marginals.Parameters = [1 1];
```

Chapter 3

Reference List

How to read the reference list

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

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

which corresponds to the following syntax:

```
Input.Name = 'My Input';
```

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

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

When one of the fields of a structure is a nested structure, a link to a table that describes the available options is provided, as in the case of the `Options` field in the following example:

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

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

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

```
Input.Option1 = 'VALUE1' ;
Input.VALUE1.Val1Opt1 = ...;
Input.VALUE1.Val1Opt2 = ...;
```

This is illustrated as follows:

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

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

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

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

3.1 Create a canonical LRA metamodel

Syntax

```
myLRA = uq_createModel(MetaOpts)
```

Input

The struct variable `MetaOpts` contains the following fields:

Table 4: `MetaOpts`

●	`.Type`	`'uq_metamodel'`	Select the metamodeling tool
●	`.MetaType`	`'LRA'`	Select low-rank approximation
●	`.Degree`	Integer	Prescribed maximum polynomial degree (no degree adaptivity)
		Integer Array	Set of polynomial degrees to consider according to `.DegSelection` (Section 2.3, Section 2.5.2.2)
●	`.Rank`	Integer	Prescribed rank (no rank adaptivity)
		Integer Array	Set of ranks to consider according to `.RankSelection` (Section 2.3, Section 2.5.2.2)
☐	`.Input`	INPUT object	Probabilistic input model (Section 2.6)
☐	`.Name`	String	Unique identifier for the metamodel
☐	`.Display`	Integer default: 1	Level of information displayed
☐	`.PolyTypes`	$1 \times M$ Cell array of strings	List of polynomial families to be used to build the LRA basis. The default choice is given in Table 3. If one of the polynomial families is Jacobi or Laguerre, the corresponding parameters should be set with `.PolyTypesParams`
☐	`.PolyTypesParams`	$1 \times M$ Cell array of doubles	Set of parameters to be used to build the LRA basis. It is only used when `.PolyTypes` contains Jacobi or Laguerre polynomials. See Section 2.5.1 for usage example
☐	`.CorrStep`	Table 5	Options relevant to the correction step (Section 2.5.2.3)
☐	`.UpdateStep`	Table 6	Options relevant to the updating step (Section 2.5.2.3)

⊞	.LRA	Table 14	Custom-LRA parameters (Section 2.9). Use the same format as the default output of the calculation
☐	.FullModel	MODEL object	UQLAB model used to create an experimental design (Section 2.6)
☐	.ExpDesign	Table 11	Experimental design-specific options (Section 2.5.3)
☐	.GenError	Table 7	Method for estimating the generalization error
☐	.RankSelection	Table 9	Rank selection options (Section 2.5.2.2)
☐	.DegSelection	Table 10	Degree selection options (Section 2.5.2.2)
☐	.Adaptivity	String default: 'all_d_adapt_r'	Adaptive strategy
		'all_d_adapt_r'	Rank adaptivity
		'all_d_adapt_d'	Degree adaptivity
		'adapt_r_d'	Rank and degree adaptivity
☐	.ValidationSet	Table 12	Validation set components (Section 2.5.4)

3.1.1 Correction step options

Full list of available options is reported in Table 5.

Table 5: MetaOpts.CorrStep

☐	.MinDerrStop	Double default: 10^{-6}	When the error decrease in two successive iterations is smaller than this value, the correction step ends
☐	.MaxIterStop	Integer default: 100	When the number of iterations reaches this value, the correction step ends
☐	.Method	String default: 'OLS'	The regression method used for the correction step
		'LARS'	Least Angle Regression
		'OLS'	Ordinary Least Squares

3.1.2 Updating step options

Full list of available options is reported in Table 6.

Table 6: MetaOpts.UpdateStep			
☐	.Method	String default: 'OLS'	The regression method used for the updating step
		'LARS'	Least Angle Regression
		'OLS'	Ordinary Least Squares

3.1.3 Error estimation options

Full list of available options is reported in Table 7.

Table 7: MetaOpts.GenError			
☐	.Method	String default: 'CV'	The method for estimating the generalization error. 'CV', corresponding to cross-validation, is the only available option in the current version of the module
☐	.Parameters	Table 8	Parameters related to GenError.Method

Table 8: MetaOpts.GenError.Parameters			
☐	.NFolds	Integer default: 3	The number of partitions of the experimental design in cross-validation

3.1.4 Rank and degree selection options

Full list of available options for rank selection is reported in Table 9.

Table 9: MetaOpts.RankSelection			
☐	.Display	Integer default: MetaOpts.Display	The local display level for the rank selection step. For display level 2, the iterations are printed
☐	.EarlyStop	Logical default: 0	Early stop for rank selection if the CV score is increasing for .EarlyStopSteps steps (Section 2.5.2.2)
☐	.EarlyStopSteps	Integer default: 2	The number of steps that the error estimator is allowed to increase before the algorithm stops

Full list of available options for degree selection is reported in Table 10.

Table 10: MetaOpts.DegSelection			
☐	.Display	Integer default: MetaOpts.Display	The local display level for the degree selection step. For display level 2, the iterations are printed
☐	.EarlyStop	Logical default: 0	Early stop for degree selection if the CV score is increasing for .EarlyStopSteps steps (Section 2.5.2.2)
☐	.EarlyStopSteps	Integer default: 2	The number of steps that the error estimator is allowed to increase before the algorithm stops

3.1.5 Experimental design

If a model is specified, UQLAB can automatically create an experimental design for LRA. The available options are listed in Table 11.

Table 11: MetaOpts.ExpDesign			
⊕	.Sampling	String default: 'MC'	Sampling type
		'MC'	Monte Carlo sampling
		'LHS'	Latin hypercube sampling
		'Sobol'	Sobol sequence sampling
		'Halton'	Halton sequence sampling
☐	.Nsamples	Integer	The number of samples to draw. It is required when .Sampling is specified
⊕	.X	$N \times M$ Double	User defined experimental design X. If specified, .Sampling is ignored
⊕	.Y	$N \times N_{out}$ Double	User defined model response Y. If specified, .Sampling is ignored
⊕	.DataFile	String	mat-file containing the experimental design. If specified, .Sampling is ignored. The mat-file must contain two variables called X and Y

3.1.6 Validation Set

If a validation set is provided, UQLAB automatically calculates the validation error of the created LRA. The required information is listed in Table 12.

Table 12: MetaOpts.ValidationSet			
●	.X	$N \times M$ Double	User-specified validation set $\boldsymbol{\mathcal{X}}_{Val}$

●	`.Y`	$N \times N_{Out}$ Double	User-specified validation set response $\mathcal{Y}_{Val}$

3.2 Accessing the results

Syntax

```
myLRA = uq_createModel(MetaOpts)
```

Output

Regardless of the configuration options specified in the `MetaOpts` structure for the construction of the metamodel, all LRA metamodels share the same output structure, described in Table 13.

Table 13: `myLRA`

`.Name`	String	Unique name of the LRA metamodel
`.Options`	Table 4	Copy of the `MetaOpts` structure used to create the metamodel
`.LRA`	Table 14	Information about all the elements of Eq. (1.4) and the metamodel mean and variance
`.ExpDesign`	Table 19	Experimental design used for developing the metamodel
`.Error`	Table 20	Error measures of the metamodel
`.Internal`	Table 21	Detailed information about the LRA computation and the internal state of the MODEL object (useful for debug/diagnostics)

3.2.1 Information on the LRA

All the information needed to evaluate and post-process the LRA metamodel are contained in the `myLRA.LRA` structure. This structure includes the LRA coefficients and moments and information on the basis, as described in the following tables.
Note that in case the considered model has an output of dimension N_{out}, each output variable Y_i is treated separately and stored in `myLRA.LRA(i)`.

Table 14: `myLRA.LRA(i)`		
`.Coefficients`	Table 15	LRA coefficients
`.Moments`	Table 16	Post-processed moments of the LRA (Section 1.4)
`.Basis`	Table 17	Information about the polynomial basis (common for all ranks)

Table 15: `myLRA.LRA(i).Coefficients`		
`.z`	$1 \times R$ cell array of $(p+1) \times M$ matrices	The univariate polynomial coefficients for each input dimension and rank
`.b`	$R \times 1$ Double	The weighing factors for each rank-one component

Table 16: `myLRA.LRA(i).Moments`		
`.Mean`	Double	Mean of the LRA (Eq. (1.21))
`.Var`	Double	Variance of the LRA (Eq. (1.22))

Table 17: `myLRA.LRA(i).Basis`		
`.Degree`	Integer	Maximum polynomial degree of the univariate polynomials
`.Rank`	Integer	Rank of the metamodel
`.PolyTypes`	$M \times 1$ cell array of strings	Polynomial family for each input variable. Each element can be one of `'Legendre'`, `'Hermite'`, `'Laguerre'` `'Jacobi'` or `'Arbitrary'`
`.PolyTypesParams`	$M \times 1$ cell array of doubles	It contains the PDF parameters when `'Jacobi'` or `'Laguerre'` polynomials are used or a `struct` that is described in Table 18
`.PolyTypesAB`	$M \times 1$ cell array	The recurrence coefficients and the region where the polynomials are computed (distribution bounds)

Table 18: `myLRA.LRA(i).Basis.PolyTypesParams`		
`.pdfname`	String	The name of the PDF used for the Stieltjes procedure
`.pdf`	Function handle	A function handle to the PDF used
`.invcdf`	Function handle	A function handle to the inverse CDF of the distribution

.cdf	Function handle	A function handle to the CDF of the distribution
.bounds	1×2 Double	The bounds of the distribution
.parameters	Vector of Doubles	The parameters of the distribution used for the Stieltjes procedure

3.2.2 Experimental design information

The experimental design and the corresponding model responses based on which the LRA metamodel is constructed are stored in the `myLRA.ExpDesign` structure. They are accessible as follows:

Table 19: `myLRA.ExpDesign`

.NSamples	Double	The number of samples
.Sampling	String	The sampling method
.ED_Input	INPUT object	The input module that represents the reduced polynomial input ($\boldsymbol{U}$ in Section 1.3.1)
.X	$N \times M$ Double	The experimental design values
.U	$N \times M$ Double	The experimental design values in the reduced space
.Y	$N \times N_{out}$ Double	The output Y that corresponds to the input X

3.2.3 Error estimates

The information relevant to the error estimates described in Section 2.5.2.1 is available in the `myLRA.Error` output field, given in Table 20.

Table 20: `myLRA.Error`

.SelectedCVScore	Double	The cross-validation error of the final LRA
.normEmpError	Double	The normalized empirical error of the final LRA
.Val	Double	Validation error (see Eq. (1.20) and Section 2.5.4). Only available if a validation set is provided (see Table 12).

3.2.4 Internal fields (advanced)

Additional information that can be useful to the advanced user is stored in the `myLRA.Internal` field. Both runtime information and complex data structures used internally by the UQLAB LRA module are stored in this structure. The general structure of the `myLRA.Internal` field is reported in Table 21. Note that not all the fields are always available, as they depend on the original configuration options.

Table 21: myLRA.Internal		
.Runtime	Table 28	Temporary variables and configuration flags used during the calculation of the LRA coefficients
.Display	Integer default: 1	The global display level that controls the level of details printed during the LRA computation
.Input	INPUT object	The probabilistic input model used to build the LRA
.FullModel	MODEL object	Full computational model used to calculate the model response (if available)
.ComputationalOpts	Table 22	Detailed information on the LRA computation parameters
.StepData	Table 25	Detailed information on the correction and updating steps performed in the construction of the final LRA

Table 22: myLRA.Internal.ComputationalOpts		
.SelectionStrategy	Table 23	The options used for the rank and degree selection
.FinalLRA	Table 24	The options used for the computation of the final LRA

Table 23: myLRA.Internal.ComputationalOpts.SelectionStrategy		
.RankSelectionOpts	Structure	Options relevant to rank selection
.DegSelectionOpts	Structure	Options relevant to degree selection
.Method	String	String specifying the adaptation strategy
.SelectionFunction	Function handle	Function handle for the adaptation strategy

Table 24: myLRA.Internal.ComputationalOpts.FinalLRA		
.CorrStep	Table 5	Correction step options
.UpdateStep	Table 6	Updating step options

Table 25: myLRA.Internal.StepData		
.CorrStep	Table 26	Information about all correction steps performed in the computation of the final LRA
.UpdateStep	Table 27	Information about all updating steps performed in the computation of the final LRA

Table 26: myLRA.Internal.StepData.CorrStep(i).Iterations(j)		
.Derr	Double array	The differential error for the j-th iteration of the i-th correction step
.errE_iter	Double array	The normalized empirical error after the j-th iteration of the i-th correction step

Table 27: myLRA.Internal.StepData.UpdateStep(i)		
.Residual	Double array	The mean (over the experimental design) residual for the i-th updating step

Table 28: myLRA.Internal.Runtime		
.isInitialized	Logical	A flag that determines whether the current metamodel has been initialized
.M	Double	The input dimension
.MnonConst	Integer	The number of non-constants in the input
.nonConstIdx	Integer array	The indices of the constant variables
.isCalculated	Logical	A flag that determines whether all the necessary quantities of the metamodel have been calculated
.Nout	Integer	The output dimension

References

Blatman, G. and Sudret, B. (2011). Adaptive sparse polynomial chaos expansion based on Least Angle Regression. *Journal of Computational Physics*, 230:2345–2367. 2

Chevreuil, M., Lebrun, R., Nouy, A., and Rai, P. (2015). A least-squares method for sparse low rank approximation of multivariate functions. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):897–921. 4, 6, 7

Deman, G., Konakli, K., Sudret, B., Kerrou, J., Perrochet, P., and Benabderrahmane, H. (2016). Using sparse polynomial chaos expansions for the global sensitivity analysis of groundwater lifetime expectancy in a multi-layered hydrogeological model. *Reliability Engineering & System Safety*, 147:156–169. 2

Doostan, A., Validi, A., and Iaccarino, G. (2013). Non-intrusive low-rank separated approximation of high-dimensional stochastic models. *Computer Methods in Applied Mechanics and Engineering*, 263:42–55. 4

Gautschi, W. (2004). *Orthogonal polynomials: computation and approximation*. Oxford University Press on Demand. 4, 18

Konakli, K. and Sudret, B. (2016a). Global sensitivity analysis using low-rank tensor approximations. *Reliability Engineering & System Safety*, 156:64 –83. 8

Konakli, K. and Sudret, B. (2016b). Polynomial meta-models with canonical low-rank approximations: Numerical insights and comparison to sparse polynomial chaos expansions. *Journal of Computational Physics*, 321:1144–1169. 4, 5, 7, 14

Lebrun, R. and Dutfoy, A. (2009). Do Rosenblatt and Nataf isoprobabilistic transformations really differ? *Probabilistic Engineering Mechanics*, 24(4):577–584. 4

Mathelin, L. (2014). Quantification of uncertainty from high-dimensional scattered data via polynomial approximation. *International Journal for Uncertainty Quantification*, 4(3):243–271. 4

Rai, P. (2014). *Sparse low rank approximation of multivariate functions – Applications in uncertainty quantification*. PhD thesis, Ecole Centrale Nantes. 4

Soize, C. and Ghanem, R. (2004). Physical systems with random uncertainties: chaos representations with arbitrary probability measure. *SIAM Journal of Scientific Computing*, 26(2):395–410. 2

Sudret, B. (2007). Uncertainty propagation and sensitivity analysis in mechanical models - Contributions to structural reliability and stochastic spectral methods. Habilitation thesis, Université Blaise Pascal, Clermont-Ferrand, France. 4

Validi, A. (2014). Low-rank separated representation surrogates of high-dimensional stochastic functions: Application in Bayesian inference. *Journal of Computational Physics*, 260:37–53. 4

Xiu, D. and Karniadakis, G. E. (2002). The Wiener-Askey polynomial chaos for stochastic differential equations. *SIAM Journal of Scientific Computing*, 24(2):619–644. 2, 4