ETH zürich

# UQLab User Manual
# Polynomial Chaos Expansions

S. Marelli, N. Lüthen, B. Sudret


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

Risk, Safety & Uncertainty Quantification

**How to cite UQLAB**

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

**How to cite this manual**

S. Marelli, N. Lüthen, B. Sudret, UQLab user manual – Polynomial chaos expansions, Report UQLab-V2.1-104, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland, 2024

**BibTeX entry**

```
@TechReport{UQdoc_21_104,
author = {Marelli, S. and L\"uthen, N. and Sudret, B.},
title = {{UQLab user manual -- Polynomial chaos expansions}},
institution = {Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich,
Switzerland},
year = {2024},
note = {Report UQLab-V2.1-104}
}
```

**List of contributors:**

| | |
|---|---|
| **M. Berchier** | **Implementation and documentation of the orthogonal matching pursuit (OMP) regression method** |
| **P. Wiederkehr** | **Implementation and documentation of the q-norm-adaptivity for regression methods** |

# Document Data Sheet

| | |
|---|---|
| Document Ref. | UQLAB-V2.1-104 |
| Title: | UQLAB user manual – Polynomial chaos expansions |
| Authors: | S. Marelli, N. Lüthen, B. Sudret<br>Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland |
| Date: | 15/04/2024 |

| Doc. Version | Date | Comments |
|---|---|---|
| V2.1 | 01/02/2022 | UQLAB V2.1 |
| V2.0 | 01/02/2022 | UQLAB V2.0 |
| V1.4 | 01/02/2021 | UQLAB V1.4 release<br>• updated the document structure<br>• added the two sparse solvers subspace pursuit and Bayesian compressive sensing<br>• added description of how to use a custom sparse solver |
| V1.3 | 19/09/2019 | UQLAB V1.3 release |
| V1.2 | 22/02/2019 | UQLAB V1.2 release<br>• updated leave-one-out error calculation<br>• added automatic calculation of validation error if a validation set is provided<br>• added q-norm adaptivity<br>• added bootstrap PCE for local error estimation |
| V1.1 | 05/07/2018 | UQLAB V1.1 release |
| V1.0 | 01/05/2017 | UQLAB V1.0 release<br>• added orthogonal matching pursuit<br>• added arbitrary polynomials support<br>• minor fixes to formatting |
| V0.9 | 01/07/2015 | Initial release |

**Abstract**

Polynomial Chaos Expansions (PCE) are a powerful metamodelling tool that has important applications in many engineering and applied mathematics fields. These include structural reliability, sensitivity analysis, Monte Carlo simulation and others. Due to the underlying complexity of its formulation, however, this technique has seen relatively little use outside of these fields.

UQLAB metamodelling tools provide an efficient, flexible and easy to use PCE module that allows one to apply state-of-the-art algorithms for non-intrusive, sparse and adaptive PCE on a variety of applications. This manual for the polynomial chaos expansion metamodelling module is divided into three parts:

- A short introduction to the main concepts and techniques behind PCE, with a selection of references to the relevant literature;
- A detailed example-based guide, with the explanation of most of the available options and methods;
- A comprehensive reference list detailing all the available functionalities in UQLAB.

**Keywords:** UQLAB, metamodelling, Polynomial Chaos Expansions, PCE, Sparse PCE

# Contents

# Chapter 1

# Theory

## 1.1 Introduction

In most modern engineering contexts (and in applied sciences in general), uncertainty quantification is becoming an increasingly important field. Deterministic scenario-based predictive modelling is being gradually substituted by stochastic modelling to account for the inevitable uncertainty in physical phenomena and measurements. This smooth transition, however, comes at the cost of dealing with greatly increased amounts of information (*e.g.* when using Monte-Carlo simulation), usually resulting in the need to perform expensive computational model evaluations repeatedly.

Metamodelling (or surrogate modelling) attempts to offset the increased costs of stochastic modelling by substituting the expensive-to-evaluate computational models (e.g. finite element models, FEM) with inexpensive-to-evaluate surrogates. Polynomial chaos expansions (PCE) are a powerful metamodelling technique that aims at providing a functional approximation of a computational model through its spectral representation on a suitably built basis of polynomial functions.

Due to the large scope of the UQLAB software framework, its polynomial chaos expansions module offers extensive facilities for the deployment of a number of non-intrusive PCE calculation techniques. This part is intended as an overview of the relevant theory and literature in this field.

## 1.2 Polynomial chaos expansion

Consider a random vector with independent components $\boldsymbol{X} \in \mathbb{R}^M$ described by the joint probability density function (PDF) $f_{\boldsymbol{X}}$. Consider also a finite variance computational model as a map $Y = \mathcal{M}(\boldsymbol{X})$, with $Y \in \mathbb{R}$ such that:

$$\mathbb{E}\left[Y^2\right] = \int_{\mathcal{D}_{\boldsymbol{X}}} \mathcal{M}^2(\boldsymbol{x}) f_{\boldsymbol{X}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} < \infty. \tag{1.1}$$

Then the polynomial chaos expansion of $\mathcal{M}(\boldsymbol{X})$ is defined as:

$$Y = \mathcal{M}(\boldsymbol{X}) = \sum_{\boldsymbol{\alpha} \in \mathbb{N}^M} y_{\boldsymbol{\alpha}} \Psi_{\boldsymbol{\alpha}}(\boldsymbol{X}) \tag{1.2}$$

where the $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X})$ are multivariate polynomials orthonormal with respect to $f_{\boldsymbol{X}}$, $\boldsymbol{\alpha} \in \mathbb{N}^M$ is a multi-index that identifies the components of the multivariate polynomials $\Psi_{\boldsymbol{\alpha}}$ and the $y_{\boldsymbol{\alpha}} \in \mathbb{R}$ are the corresponding coefficients (coordinates).
In realistic applications, the sum in Eq. (1.2) needs to be truncated to a finite sum, by introducing the truncated polynomial chaos expansion:

$$\mathcal{M}(\boldsymbol{X}) \approx \mathcal{M}^{PC}(\boldsymbol{X}) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}} y_{\boldsymbol{\alpha}} \Psi_{\boldsymbol{\alpha}}(\boldsymbol{X}) \tag{1.3}$$

where $\mathcal{A} \subset \mathbb{N}^M$ is the set of selected multi-indices of multivariate polynomials. The construction of such a set is detailed in Section 1.3.3.

## 1.3 Building the polynomial basis

The polynomial basis $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X})$ in Eq. (1.3) is traditionally built starting from a set of *univariate orthonormal polynomials* $\phi_k^{(i)}(x_i)$ which satisfy:

$$\left\langle \phi_j^{(i)}(x_i), \phi_k^{(i)}(x_i) \right\rangle \overset{\text{def}}{=} \int_{\mathcal{D}_{X_i}} \phi_j^{(i)}(x_i) \phi_k^{(i)}(x_i) f_{X_i}(x_i) \mathrm{d}x_i = \delta_{jk} \tag{1.4}$$

where $i$ identifies the input variable w.r.t. which they are orthogonal as well as the corresponding polynomial family, $j$ and $k$ the corresponding polynomial degree, $f_{X_i}(x_i)$ is the $i^{th}$-input marginal distribution and $\delta_{jk}$ is the Kronecker symbol. Note that this definition of inner product can be interpreted as the expectation value of the product of the factors.

The multivariate polynomials $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X})$ are then assembled as the tensor product of their univariate counterparts:

$$\Psi_{\boldsymbol{\alpha}}(\boldsymbol{x}) \overset{\text{def}}{=} \prod_{i=1}^{M} \phi_{\alpha_i}^{(i)}(x_i) \tag{1.5}$$

Due to the orthonormality relations in Eq. (1.4), it follows that also the multivariate polynomials thus constructed are orthonormal:

$$\langle \Psi_{\boldsymbol{\alpha}}(\boldsymbol{x}), \Psi_{\boldsymbol{\beta}}(\boldsymbol{x}) \rangle = \delta_{\boldsymbol{\alpha}\boldsymbol{\beta}} \tag{1.6}$$

where $\delta_{\boldsymbol{\alpha}\boldsymbol{\beta}}$ is an extension Kronecker symbol to the multi-dimensional case.

### 1.3.1 Classical families of univariate orthonormal polynomials

The classical families of univariate orthonormal polynomials and the distributions to which they are orthonormal are given for reference in Table 1 (Sudret, 2007). Detailed descriptions of each of the classical families of polynomials (also called Askey-Scheme orthonormal

Table 1: List of classical univariate polynomial families common in polynomial chaos expansion applications.

| 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 $J_k^{a,b}(x)$ | $J_k^{a,b}(x)/\mathfrak{J}_{a,b,k}$ |
| | | $\mathfrak{J}^2_{a,b,k} = \frac{2^{a+b+1}}{2k+a+b+1}\frac{\Gamma(k+a+1)\Gamma(k+b+1)}{\Gamma(k+a+b+1)\Gamma(k+1)}$ | |

polynomials) given in the table are abundant in the literature, see *e.g.* Xiu and Karniadakis (2002). The values of the polynomials are computed using Equation 1.9.

**Note:** The computation of the polynomial values via recurrence relation is not always stable. In practice, Laguerre and Jacobi polynomials of degree over 23 should be avoided. See Gautschi (1993) for details.

### 1.3.2 Extension to arbitrary distributions

In case the input random variables are not independent, or no standard polynomials are defined for their marginal distributions, there are two possibilities for the choice of marginal distributions: performing an isoprobabilistic transform to a space with independent standard marginals, or computation of a custom set of polynomials which are orthonormal to the non-standard distribution.

#### 1.3.2.1 Isoprobabilistic transform

It is possible to define an *isoprobabilistic transform* from the original probabilistic space to the so-called *reduced space*, which has the same dimensionality, but where the input random variables are independent and have standard marginals. Consider an input vector of random variables $\boldsymbol{Z}$ with joint PDF $\boldsymbol{Z} \sim f_{\boldsymbol{Z}}(\boldsymbol{z})$. Then, there exists an isoprobabilistic transform $\mathcal{T}$ such that:

$$\boldsymbol{X} = \mathcal{T}(\boldsymbol{Z}), \qquad \boldsymbol{Z} = \mathcal{T}^{-1}(\boldsymbol{X}) \tag{1.7}$$

where $\boldsymbol{X}$ is a random vector with independent components distributed according to one of the distributions in Table 1. We can then compute a PCE in terms of $\boldsymbol{X}$ as in Eq. (1.2) and rewrite it in terms of $\boldsymbol{Z}$ as follows:

$$Y = \mathcal{M}(\boldsymbol{Z}) = \sum_{\boldsymbol{\alpha}\in\mathbb{N}^M} y_{\boldsymbol{\alpha}} \Psi_{\boldsymbol{\alpha}} \left(\mathcal{T}\left(\boldsymbol{Z}\right)\right) \tag{1.8}$$

This transform also allows to use any type of orthonormal polynomial with any type of input marginals, at the cost of an additional isoprobabilistic transform. Note that this type of transform can be highly non-linear, more so when transforming from a compact to a non-

compact support distribution (e.g., uniform to Gaussian). The added non-linearity can have a significant detrimental effect on the accuracy of the final truncated PCE, because it may result in a more complex model.
For simplicity and without loss of generality, we will leave out any reference to possible isoprobabilistic transforms in the following and consider the random vector $\boldsymbol{X}$ as the vector of independent input variables with marginal distributions defined in Table 1.

#### 1.3.2.2 Polynomials orthonormal with respect to arbitrary distributions

For a given random variable with probability density function $f(x)$, it is possible to construct an associated set of univariate polynomials $\pi_n, n = 0, 1, 2, \ldots$, which are orthogonal to $f(x)$ (provided certain mild assumptions on $f(x)$ hold, see Ernst et al. (2012)). The corresponding orthonormal polynomials are given by $\tilde{\pi}_n = \frac{\pi_n}{\sqrt{\langle \pi_n, \pi_n \rangle}}$, where $\langle g, h \rangle = \int g(x)h(x)f(x)dx$.
In UQLAB, the polynomials are computed through the *Stieltjes procedure*. This procedure generates a sequence of univariate polynomials, which are of increasing degree and orthogonal to the given probability distribution $f(x)$, using the following recurrence relation:

$$\sqrt{\beta_{n+1}}\tilde{\pi}_{n+1}(x) = (x - \alpha_n)\tilde{\pi}_n(x) - \sqrt{\beta_n}\tilde{\pi}_{n-1}(x), \quad n = 0, 1, 2, \ldots \tag{1.9}$$

where $\alpha_n$ and $\beta_n$ are defined by the so-called *Christoffel-Darboux* formulae

$$\alpha_n = \frac{\langle x\pi_n, \pi_n \rangle}{\langle \pi_n, \pi_n \rangle}, \tag{1.10}$$

$$\beta_n = \frac{\langle \pi_n, \pi_n \rangle}{\langle \pi_{n-1}, \pi_{n-1} \rangle}. \tag{1.11}$$

The numerical integrations for the inner products involved in the Christoffel-Darboux formulae are performed using the MATLAB adaptive integrator (Shampine (2008)). For more details on the procedure refer to Gautschi (2004).

### 1.3.3 Basis truncation schemes

Given the polynomials in Table 1, it is straightforward to define the *total-degree truncation scheme*, which corresponds to all polynomials in the $M$ input variables of total degree less than or equal to $p$:

$$\mathcal{A}^{M,p} = \{\boldsymbol{\alpha} \in \mathbb{N}^M \ : \ |\boldsymbol{\alpha}| \leq p\} \qquad \text{card } \mathcal{A}^{M,p} \equiv P = \binom{M+p}{p} \tag{1.12}$$

Note that the total-degree basis grows exponentially with the degree $p$.
In many applied science problems, not all terms in the basis are equally important. Often, the important terms in the expansion tend to be the ones where only few variables are involved. This is known as the *sparsity-of-effects principle*. Based on this reasoning, two additional truncation schemes have been proposed which have the effect of excluding terms of high interaction order.

#### 1.3.3.1 Restriction of maximum interaction

This truncation scheme is based on choosing a subset of the terms defined in Eq. (1.12), such that the $\boldsymbol{\alpha}$'s have at most $r$ non-zero elements (low-rank $\boldsymbol{\alpha}$):

$$\mathcal{A}^{M,p,r} = \{\boldsymbol{\alpha} \in \mathcal{A}^{M,p} \ : \ ||\boldsymbol{\alpha}||_0 \leq r\}, \tag{1.13}$$

where $||\boldsymbol{\alpha}||_0 = \sum_{i=1}^{M} \mathbf{1}_{\{\alpha_i > 0\}}$ is the *rank* of the multi-index $\boldsymbol{\alpha}$. For a multivariate polynomial $\psi_{\boldsymbol{\alpha}}(\boldsymbol{x})$, the rank of $\boldsymbol{\alpha}$ corresponds to the number of variables involved in the polynomial, also called *interaction order*.

This truncation scheme can be used to significantly reduce the cardinality of the polynomial basis by limiting the number of interaction terms, which is particularly effective in high dimension.

#### 1.3.3.2 Hyperbolic truncation

A modification of the standard scheme, the hyperbolic (or *q-norm*) truncation scheme makes use of the so-called *q-norm* to define the truncation (Blatman, 2009):

$$\mathcal{A}^{M,p,q} = \{\boldsymbol{\alpha} \in \mathcal{A}^{M,p} \ : \ ||\boldsymbol{\alpha}||_q \leq p\}, \tag{1.14}$$

where:

$$||\boldsymbol{\alpha}||_q = \left(\sum_{i=1}^{M} \alpha_i^q\right)^{1/q}. \tag{1.15}$$

Note that for $q = 1$, hyperbolic truncation corresponds exactly to the standard total-degree truncation scheme in Eq. (1.12). For $q < 1$, hyperbolic truncation includes all the univariate high-degree terms, but excludes high-degree terms with many interacting variables. An example of the behaviour of the hyperbolic norm in two dimensions for different values of $p$ and $q$ is shown in Figure 1.

### 1.3.4 Basis-adaptive PCE

In real-world applications, it is often not clear which finite basis would yield the best PCE approximation. On one hand, the basis must contain enough elements to enable an accurate representation. On the other hand, the available number of experimental design points limits the size of the basis.

In this case, a popular strategy is *basis-adaptive PCE*, where a suitable basis is chosen from a set of candidate bases. Here, starting from a small candidate basis, one gradually generates new bases by adding new elements (*e.g.* by increasing the maximum polynomial degree in the truncation scheme) and calculates the corresponding PCE and (an approximation to) its generalization error. Finally, the best PCE in terms of generalization error is chosen. For any adaptive algorithm, a crucial tool is *a-posteriori* cross-validation error estimation (see

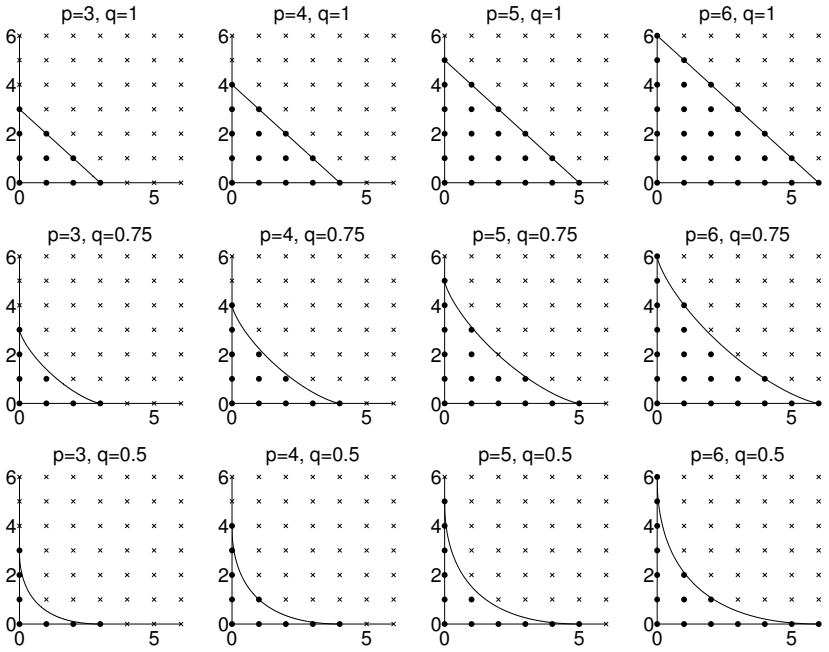



Figure 1: Hyperbolic truncation set for varying values of $p$ (constant in each column) and $q$ (constant in each row) as defined in Eq. (1.14). For $q = 1$, hyperbolic truncation reduces to the standard total-degree truncation scheme (first row). Decreasing the value of $q$ decreases the number of polynomials of high interaction order included in the expansion.

Section 1.4) because it allows one to estimate the accuracy of the model without having to run additional expensive model evaluations to generate a proper validation set.

Two commonly used strategies for basis-adaptive PCE, which can also be combined with each other, are *degree adaptivity* and *q-norm adaptivity*. Given a target accuracy $\epsilon_T$ and a maximum number of iterations $NI_{max}$, the algorithms can be summarized as follows:

1. Generate an initial basis with one or a combination of the truncation schemes in Section 1.3.3, with $p = p_0$ ($q = q_0$);

2. Calculate the PCE coefficients and the corresponding generalization error estimate $\epsilon$ (*e.g.* $\epsilon_{LOO}$ in Eqs. (1.19) or (1.22));

3. Compare the error with the preset threshold $\epsilon_T$. If $\epsilon \leq \epsilon_T$ or the number of iterations $NI = NI_{max}$, stop the algorithm and return the PCE with the lowest generalization error. Otherwise, set $p = p + 1$ (increase $q$) and return to Step 1.

This simple algorithm is effective in letting the maximum degree or the q-norm of the PCE be driven directly from the available data. For this type of algorithm to properly converge, an error estimate sensitive to over-fitting should be chosen, *e.g.* $\epsilon_{LOO}$.

## 1.4 *A posteriori* error estimation

After the metamodel is constructed (see Section 1.5), its accuracy and predictive quality can be quantified by estimating the relative generalization error $\epsilon_{gen}$. It is defined as:

$$\epsilon_{gen} = \frac{\mathbb{E}\left[\left(\mathcal{M}(\boldsymbol{X}) - \mathcal{M}^{PC}(\boldsymbol{X})\right)^2\right]}{\mathrm{Var}\left[Y\right]} \tag{1.16}$$

In case next to the training set used to construct the metamodel an independent set of inputs and outputs, also called *validation set*, $[\boldsymbol{\mathcal{X}}_{Val},\ \mathcal{Y}_{Val} = \mathcal{M}(\boldsymbol{\mathcal{X}}_{Val})]$ is available, the *validation error* can be calculated as:

$$\epsilon_{Val} = \frac{\sum_{i=1}^{N}\left(\mathcal{M}(\boldsymbol{x}_{Val}^{(i)}) - \mathcal{M}^{PC}(\boldsymbol{x}_{Val}^{(i)})\right)^2}{\sum_{i=1}^{N}\left(\mathcal{M}(\boldsymbol{x}_{Val}^{(i)}) - \hat{\mu}_{Y_{Val}}\right)^2} \tag{1.17}$$

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 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 $\epsilon_{gen}$: *normalized empirical error* and *leave-one-out cross-validation error*.

### 1.4.1 Normalized empirical error

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

$$\epsilon_{emp} = \frac{\sum_{i=1}^{N}\left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \mathcal{M}^{PC}(\boldsymbol{x}^{(i)})\right)^2}{\sum_{i=1}^{N}\left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \hat{\mu}_Y\right)^2} \tag{1.18}$$

where $\hat{\mu}_Y$ is the sample mean of the experimental design response.
The estimator in Eq. (1.18), although inexpensive to calculate, leads to over-fitting: it is a monotone decreasing function of the polynomial degree $p$ regardless of the size of the experimental design.

### 1.4.2 Leave-one-out cross-validation error

The leave-one-out (LOO) cross-validation error $\epsilon_{LOO}$ is designed to overcome the over-fitting limitation of $\epsilon_{emp}$ by using cross-validation, a technique developed in statistical learning theory. It consists in building $N$ metamodels $\mathcal{M}^{PC\setminus i}$, each one created on a reduced experimental design $\boldsymbol{\mathcal{X}}\setminus\boldsymbol{x}^{(i)} = \{\boldsymbol{x}^{(j)}, j = 1, ..., N, j \neq i\}$ and comparing its prediction on the excluded

point $\boldsymbol{x}^{(i)}$ with the real value $y^{(i)}$ (see, e.g., Blatman and Sudret (2010)). The leave-one-out cross-validation error can be written as:

$$\epsilon_{LOO} = \frac{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \mathcal{M}^{PC\setminus i}(\boldsymbol{x}^{(i)})\right)^2}{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \hat{\mu}_Y\right)^2}. \tag{1.19}$$

In practice, when the results of a least-square minimization (see Section 1.5.2) are available, there is no need to explicitly calculate $N$ separate metamodels, but one can use the following formulation to calculate $\epsilon_{LOO}$ (see Blatman (2009), appendix D):

$$\epsilon_{LOO} = \frac{\sum_{i=1}^{N} \left(\frac{\mathcal{M}(\boldsymbol{x}^{(i)}) - \mathcal{M}^{PC}(\boldsymbol{x}^{(i)})}{1-h_i}\right)^2}{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \hat{\mu}_Y\right)^2}, \tag{1.20}$$

where $h_i$ is the $i^{th}$ component of the vector given by:

$$\boldsymbol{h} = \text{diag}\left(\mathbf{A}(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}\mathbf{A}^\mathsf{T}\right), \tag{1.21}$$

and $\mathbf{A}$ is the experimental matrix in Eq. (1.32).

### 1.4.3 Corrected error estimates

Further empirical corrections exist to ensure that the generalization error estimates from both $\epsilon_{emp}$ and $\epsilon_{LOO}$ are not underestimated. They generally take the form:

$$\epsilon^* = \epsilon\, T(P, N), \tag{1.22}$$

where $\epsilon^*$ is the corrected error, $\epsilon$ is the original error and $T(P, N)$ is a correction factor that typically increases with the number of regressors $P$ and tends to 1 when the size of the experimental design $N \to \infty$.
A correction factor particularly effective in the context of PCE with small experimental designs (Blatman (2009), after Chapelle et al. (2002)) is given by:

$$T(P, N) = \frac{N}{N-P}\left(1 + \frac{\text{tr}(\mathbf{C}_{emp}^{-1})}{N}\right), \tag{1.23}$$

with

$$\mathbf{C}_{emp} = \frac{1}{N}\mathbf{A}^\mathsf{T}\mathbf{A}. \tag{1.24}$$

## 1.5 Calculation of the coefficients

Several methods exist to calculate the coefficients $y_{\boldsymbol{\alpha}}$ of the polynomial chaos expansion for a given basis (Eq (1.3)). In UQLAB, only non-intrusive methods are implemented, *i.e.* the

coefficients are obtained by post-processing of the *experimental design*, a set consisting of samples of the input random variables and the corresponding model evaluations.
The two principal strategies to calculate the polynomial chaos coefficients non-intrusively are *projection* and *regression*. Projection methods use the orthogonality of the basis functions to compute the coefficients by numerical integration (quadrature). Regression methods formulate Eq. (1.2) as a system of linear equations and solve the system by standard linear regression approaches. One such regression approach is *ordinary least-squares regression* (Section 1.5.2), for which the system of linear equations must be overdetermined. Alternatively, *sparse regression* techniques can be used, which aim to find coefficient vectors with only few nonzero entries (*i.e.*, *sparse* solutions), and which are able to find solutions to underdetermined systems of equations.

### 1.5.1 Projection method

The calculation of the polynomial coefficients $y_{\boldsymbol{\alpha}}$ with the projection approach directly follows from the definition of PCE given in Eq. (1.2) and from the orthonormality of the polynomial basis. Indeed, taking the expectation value of Eq. (1.2) multiplied by $\Psi_{\boldsymbol{\beta}}(\boldsymbol{x})$ yields:

$$y_{\boldsymbol{\alpha}} = \mathbb{E}\left[\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X}) \cdot \mathcal{M}(\boldsymbol{X})\right] \tag{1.25}$$

The calculation of the coefficients is therefore reduced to the calculation of the expectation value in Eq. (1.25). It can be cast as a numerical integration problem which in turn can be efficiently solved using quadrature methods.

**Gaussian quadrature:** a standard tool in the numerical evaluation of integrals, Gaussian quadrature is based on a simple weighted-sum scheme:

$$y_{\boldsymbol{\alpha}} = \int_{\Omega_{\boldsymbol{X}}} \mathcal{M}(\boldsymbol{x})\Psi_{\boldsymbol{\alpha}}(\boldsymbol{x})f_{\boldsymbol{X}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \approx \sum_{i=1}^{N} w^{(i)}\mathcal{M}(\boldsymbol{x}^{(i)})\Psi_{\boldsymbol{\alpha}}(\boldsymbol{x}^{(i)}) \tag{1.26}$$

The set of weights $w^{(i)}$ and quadrature points $x^{(i)}$ (the experimental design) are derived from Lagrange polynomial interpolation and guarantees exactness in the evaluation of the integrals of functions of polynomial complexity (Gander and Gautschi (2000)).
The integration weights $w^{(i)}$ and the integration nodes $\boldsymbol{x}^{(i)}$ are uniquely determined by the marginals of the independent components of the input random vector $\boldsymbol{X}$, and they correspond to the roots of the corresponding polynomial basis functions as reported in Table 1. In UQLAB the Gaussian quadrature nodes and weights are numerically calculated with the Golub-Welsch algorithm (Gautschi (2004), Golub and Welsch (1969)).
Standard multivariate Gaussian quadrature is achieved by a tensor-product of univariate integration rules. Therefore the number of integration nodes (*i.e.* full-model evaluations) increases rapidly with the number of input variables. As an example, selecting a max polynomial degree $p$ would require $(p+1)$ integration points in each dimension, leading to $N = (p+1)^M$ in Eq (1.26). This is the so-called curse of dimensionality.

**Sparse quadrature:** a more recent tool to deal with high-dimensional integration, Smolyak' sparse quadrature is an alternative approach to the original tensor-product multi-dimensional quadrature (Gerstner and Griebel, 1998). The integration is still performed as given in Eq. (1.26), but the weights are derived from a combination of lower order standard quadrature terms:

$$Q_{Smolyak}^{M,l} \equiv \sum_{l+1 \leqslant |\boldsymbol{i}| \leqslant l+M} (-1)^{M+l-|\boldsymbol{i}|} \cdot \binom{M-1}{k+M-|\boldsymbol{i}|} \cdot Q^{\boldsymbol{i}} \tag{1.27}$$

where:

$$\boldsymbol{i} = i_1, i_2, ..., i_M, \qquad |\boldsymbol{i}| = i_1 + ... + i_M \in \mathbb{N}$$

and

$$Q^{\boldsymbol{i}} = Q^{i_1} \otimes .... \otimes Q^{i_M}$$

is a tensor product of the lower order Gaussian quadrature rules identified by the multi-index $\boldsymbol{i}$. This method can lead to a substantially reduced number of integration points w.r.t. classical Gaussian quadrature without sacrificing accuracy in higher dimensions.

#### 1.5.1.1 Error estimation

An *a posteriori* error estimate of the Gaussian quadrature error in the estimation of the PCE coefficients in Eq. (1.26) can be calculated by taking the expectation value of the residual mean-square error $\mathbb{E}\left[\mathcal{M}(X) - \mathcal{M}^{PC}(X)\right]$ by integrating it with the same quadrature rule and on the same nodes:

$$\epsilon_{res} \approx \frac{\sum_{i=1}^{N} \left[w^{(i)} \left(Y^{(i)} - \boldsymbol{y}^{\mathsf{T}} \boldsymbol{\Psi}(\boldsymbol{x}^{(i)})\right)\right]^2}{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \hat{\mu}_Y\right)^2} \tag{1.28}$$

where $\boldsymbol{y} = \{y_{\boldsymbol{\alpha}_1}, \ldots, y_{\boldsymbol{\alpha}_P}\}^{\mathsf{T}}$ is the vector of polynomial coefficients, $\boldsymbol{\Psi}(\boldsymbol{x}^{(i)}) = \left\{\Psi_{\boldsymbol{\alpha}_1}(\boldsymbol{x}^{(i)}), \ldots, \Psi_{\boldsymbol{\alpha}_P}(\boldsymbol{x}^{(i)})\right\}^{\mathsf{T}}$ is a vector containing the values of the polynomial basis elements at quadrature point $\boldsymbol{x}^{(i)}$ and $\hat{\mu_Y} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{M}(\boldsymbol{x}^{(i)})$ is the sample mean of the set of quadrature points.

### 1.5.2 Least-Squares regression

A different approach to estimate the coefficients in Eq. (1.3) is to set up a least-squares minimization problem (Berveiller et al., 2006). The infinite series in Eq. (1.2) can be written as a sum of its truncated version Eq. (1.3) and a residual:

$$Y = \mathcal{M}(\boldsymbol{X}) = \sum_{j=0}^{P-1} y_j \, \Psi_j(\mathbf{X}) + \varepsilon_P \equiv \boldsymbol{y}^{\mathsf{T}} \Psi(\boldsymbol{X}) + \varepsilon_P \tag{1.29}$$

where $P = \text{card}\, \mathcal{A}^{M,p}$, $\epsilon_P$ is the truncation error, $\boldsymbol{y}_{\boldsymbol{\alpha}} = \{y_0, \ldots, y_{P-1}\}^{\mathsf{T}}$ is a vector containing the coefficients and $\Psi(\boldsymbol{X}) = \{\Psi_0(\boldsymbol{X}), \ldots, \Psi_{P-1}(\boldsymbol{X})\}^{\mathsf{T}}$ is the vector that assembles the

values of all the orthonormal polynomials in $\boldsymbol{X}$.
The least-square minimization problem can then be set-up as:

$$\hat{\boldsymbol{y}} = \arg\min \mathbb{E}\left[\left(\boldsymbol{y}^{\mathsf{T}}\Psi(\boldsymbol{X}) - \mathcal{M}(\boldsymbol{X})\right)^2\right]. \tag{1.30}$$

#### 1.5.2.1 Ordinary Least-Squares

A direct approach to solving Eq. (1.30) is given by Ordinary Least-Squares (OLS). Given a sample $\boldsymbol{\mathcal{X}} = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}\}^{\mathsf{T}}$ of size $N$ of the input random vector $\boldsymbol{X}$ (the experimental design) and the corresponding model responses $\mathcal{Y} = \{y^{(1)}, ..., y^{(N)}\}^{\mathsf{T}}$, the ordinary least-squares solution of Eq. (1.30) reads:

$$\hat{\boldsymbol{y}} = (\mathbf{A}^{\mathsf{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathsf{T}}\mathcal{Y}, \tag{1.31}$$

where

$$A_{ij} = \Psi_j\left(\boldsymbol{x}^{(i)}\right) \quad i = 1, \ldots, n\,;\; j = 0, \ldots, P-1 \tag{1.32}$$

is the so-called *experimental matrix* (or *regression matrix*) that contains the values of all the basis polynomials in the experimental design points.
The main advantage of the least-squares minimization method over the projection method lies in the fact that an arbitrary number of points can be used to calculate the coefficients, as long as they are a representative sample of the random input vector $\boldsymbol{X}$ and $N \geq P$. A theoretical analysis of the convergence of the least-squares minimization method can be found in Migliorati et al. (2013).

### 1.5.3 Sparse PCE: Least Angle Regression

A complementary strategy to favour sparsity in high dimension consists in directly modifying the least-square minimization problem in Eq. (1.30) by adding a penalty term of the form $\lambda||\boldsymbol{y}||_1$, *i.e.* solving:

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y}\in\mathbb{R}^{|\mathcal{A}|}}{\operatorname{argmin}}\, \mathbb{E}\left[\left(\boldsymbol{y}^{\mathsf{T}}\Psi(\boldsymbol{X}) - \mathcal{M}(\boldsymbol{X})\right)^2\right] + \lambda||\boldsymbol{y}||_1, \tag{1.33}$$

where the regularization term $||\hat{\boldsymbol{y}}||_1 = \sum_{\boldsymbol{\alpha}\in\mathcal{A}} |y_{\boldsymbol{\alpha}}|$ forces the minimization to favour low-rank solutions. Several algorithms exist that solve the penalized minimization problem in Eq. (1.33), including least absolute shrinkage and selection operator (LASSO, Tibshirani (1996)), forward stagewise regression (Hastie et al., 2007) and least angle regression, or LAR, (Efron et al., 2004). In the context of PCE, Blatman and Sudret (2011) successfully applied the LAR algorithm to obtain sparse PCE models that are accurate even with very small experimental designs.

#### 1.5.3.1 The LAR algorithm

The LAR algorithm is a linear regression tool based on iteratively moving regressors from a *candidate set* to an *active set*. The next regressor is chosen based on its correlation with the

current residual. At each iteration, analytical relations are used to identify the best set of regression coefficients for that particular active set, by imposing that every active regressor is equicorrelated with the current residual.
The optimal number of predictors in the metamodel (i.e. the optimal number of LAR steps) may be determined using a suitable criterion.
The full LAR algorithm in the context of PCE (Blatman and Sudret, 2011) reads:
**Initialization**

- $y_{\boldsymbol{\alpha}} = 0, \qquad \forall \boldsymbol{\alpha} \in \mathcal{A}^{M,p,q}$;
- Candidate set: $\Psi_{\boldsymbol{\alpha}}$;
- Active set: $\varnothing$;
- Residual: $r_0 = \mathcal{Y}$

**Iterative algorithm:**

1. Find the regressor $\Psi_{\boldsymbol{\alpha}_j}$ that is most correlated with the current residual
2. Move all the coefficients of the current active set towards their least-square value until their regressors are equicorrelated to the residual as some other regressor in the candidate set. This regressor will also be the most correlated to the residual in the next iteration.
3. Calculate and store the error estimate $\epsilon_{LOO}^j$ for the current iteration
4. Update all the active coefficients and move $\boldsymbol{\Psi}_{\boldsymbol{\alpha}_j}$ from the candidate set to the active set
5. Repeat the previous steps until the size of the active set is equal to $m = \min(P, N-1)$

After the iterations are finished, the candidate set of regressors with the lowest $\epsilon_{LOO}$ is selected as the best sparse candidate basis. A typical example of the evolution of $\epsilon_{LOO}^j$ vs. $j$ is shown in Figure 2.

#### 1.5.3.2 Hybrid LAR

One limitation of the LAR algorithm is that it is defined only for non-constant regressors (due to the presence of the cross-correlation-based selection at the first step of the algorithm). Moreover, the hyperparameter of the algorithm is the number of LAR steps. Therefore, the *leave-one-out error* can not be calculated as easily as OLS that does not require rebuilding $N$ models (see Eq. (1.20)). Both limitations can be overcome by introducing the so-called *hybrid-LAR* step. At the end of each LAR iterations, the constant regressor is added to the selected basis, and OLS is performed to calculate the related coefficients as well as the corresponding *leave-one-out error*. Thus, in this setting, LAR iterations are used to provide a series of set of basis functions, and OLS is used to build the associated surrogate models. The final model is chosen as the one having the smallest *leave-one-out error*.

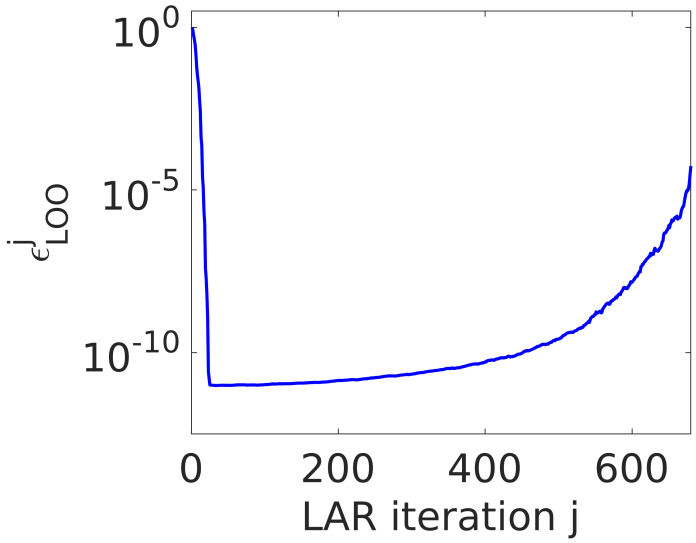



Figure 2: Typical evolution of $\epsilon_{LOO}$ vs. LAR iteration. The evolution is in many cases smooth and consistently convex throughout the iterations. In some cases with very small experimental designs, however, local minima in the early iterations can be observed.

#### 1.5.3.3 LAR early stop criterion

When dealing with a large number of regressors, each LAR iteration can be time consuming (mostly due to the calculation of the $\epsilon_{LOO}$, which entails a large matrix inversion). An early stop criterion can be introduced in practical implementations to mitigate the corresponding costs and speed-up the algorithm. The criterion stems from the observation that in most real-case scenarios the behaviour of $\epsilon^j_{LOO}$ is relatively smooth with the iteration number $j$ (see Figure 2) and convex. An effective and robust early stop criterion for LAR is then to stop adding regressors after the $\epsilon_{LOO}$ is above its minimum value for at least $10\%$ of the maximum number of possible iterations.

### 1.5.4 Sparse PCE: Orthogonal Matching Pursuit

Orthogonal Matching Pursuit (OMP) is a greedy algorithm proposed by Pati et al. (1993) as a refinement to the Matching Pursuit algorithm by Mallat and Zhang (1993). OMP works by iteratively retrieving the polynomial basis elements that are most correlated with the current approximation residual and adding them to the *active set* of regressors.

OMP uses a greedy iterative strategy that minimizes the approximation residual at each iteration. Consider the approximation residual $R_n$ for a polynomial basis with $n$ elements and the approximation residual $R_{n+1}$ for a polynomial basis with $n+1$ elements. By projecting the residual $R_n$ onto a new polynomial the following equation is obtained:

$$R_n = \left\langle R_n, \Psi_{\boldsymbol{\alpha}_{n+1}} \right\rangle \Psi_{\boldsymbol{\alpha}_{n+1}} + R_{n+1} \tag{1.34}$$

Therefore, the residual $R_{n+1}$ is by construction orthogonal to the new polynomial $\Psi_{\boldsymbol{\alpha}_{n+1}}$.

Projecting Eq. (1.34) onto $R_n$ yields:

$$\left\|R_n\right\|^2 = \left|\langle R_n, \Psi_{\boldsymbol{\alpha}_{n+1}}\rangle\right|^2 + \left\|R_{n+1}\right\|^2. \tag{1.35}$$

It follows that minimizing the approximation residual $R_{n+1}$ is equivalent to choosing a polynomial such that $\left|\langle R_n, \Psi_{\boldsymbol{\alpha}_{n+1}}\rangle\right|$ is maximized. Therefore, each iteration of the OMP algorithm consists in solving the following problem:

$$\Psi_{\boldsymbol{\alpha}_{n+1}} = \arg\max_{\boldsymbol{\alpha}\in\mathcal{A}} \left|\langle R_n, \Psi_{\boldsymbol{\alpha}}\rangle\right| \tag{1.36}$$

After the basis element $\Psi_{\boldsymbol{\alpha}_{n+1}}$ has been added to the *active set* of regressors, all the corresponding polynomial coefficients $y_{\boldsymbol{\alpha}}$ are updated via ordinary least squares. This additional step guarantees that the newly calculated residual is orthogonal to *all* the regressors in the current *active set*.

#### 1.5.4.1 The OMP algorithm

The OMP algorithm is a linear regression tool that minimizes the norm of the approximation residual at each iteration. The algorithm uses the *leave-one-out error* estimator in Eq. (1.19) to adaptively select the best active set. The implementation of the OMP algorithm in the context of PCE reads:

**Initialization:**

- $y^0_{\boldsymbol{\alpha}} = 0, \qquad \forall \boldsymbol{\alpha} \in \mathcal{A}^{M,p,q}$;
- Candidate set: $\boldsymbol{\Psi}_{C,0} = \Psi_{\boldsymbol{\alpha}}$;
- Active set: $\boldsymbol{\Psi}_{A,0} = \varnothing$;
- Residual: $R_0 = \mathcal{Y}$

**Iterative algorithm:**

1. Find the polynomial $\Psi_{\boldsymbol{\alpha}_j}$ that is most correlated with the current approximation residual $R_{j-1}$.
2. Add the polynomial $\Psi_{\boldsymbol{\alpha}_j}$ to the active set of polynomials, i.e. $\boldsymbol{\Psi}_{A,j} = \boldsymbol{\Psi}_{A,j-1} \cup \Psi_{\boldsymbol{\alpha}_j}$.
3. Calculate the new polynomial coefficients $y^j_{\boldsymbol{\alpha}}$ by projecting the model response $\mathcal{Y}$ onto the active set of polynomials, i.e. calculate an ordinary least squares using the active set.
4. Calculate the new approximation residual $R_j = \mathcal{Y} - \boldsymbol{\Psi}_{A,j} y^j_{\boldsymbol{\alpha}}$.
5. Calculate and store the error estimate $\epsilon^j_{LOO}$ for the current iteration.
6. Repeat the previous steps until the size of the active set is equal to $m = \min(P, N)$.

After the iterative procedure is terminated, the active set of polynomials with the lowest $\epsilon_{LOO}$ is selected as the best sparse basis.

#### 1.5.4.2 OMP early stop criterion

When the polynomial basis size $P$ is large, each OMP iteration can be computationally expensive. An early stop criterion can be used to reduce the computational costs. Similar to the LAR algorithm, the behaviour of $\epsilon_{LOO}^{j}$ is relatively smooth and convex (see Figure 3). The proposed early stop criterion for OMP stops adding regressors after the $\epsilon_{LOO}$ is above its minimum value for at least 10% of the maximum number of possible iterations.

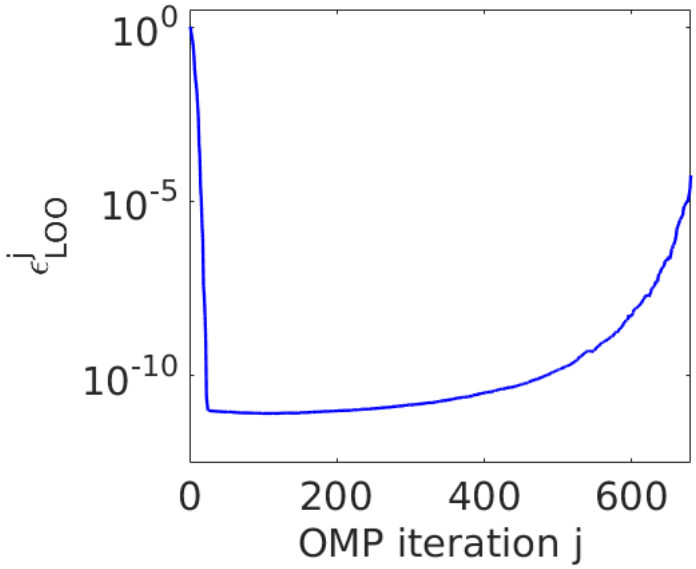



Figure 3: Typical evolution of $\epsilon_{LOO}$ vs. OMP iterations. The evolution is in many cases relatively smooth and consistently convex throughout the iterations. In some cases with very small experimental designs, however, local minima in the early iterations can be observed.

### 1.5.5 Sparse PCE: Subspace Pursuit

Subspace pursuit (SP) is a sparse regression algorithm developed by Dai and Milenkovic (2009) and introduced for PCE by Diaz et al. (2018). Its single hyperparameter is the number $K$ of nonzero elements in the coefficient vector. Starting from an initial set of $K$ regressors that are highly correlated with the model evaluations, it computes the corresponding coefficients by ordinary least-squares (OLS), identifies $K$ more regressors that are highly correlated with the residual, computes the OLS solution on the combined set of $2K$ regressors, and removes again the $K$ regressors with the smallest-in-magnitude coefficients. This is repeated until convergence.

Theoretical guarantees for this algorithm were derived by Dai and Milenkovic (2009). Numerically, it has been found that SP performs particularly well for low-dimensional problems with moderate to large experimental design sizes (Lüthen et al., 2020).

#### 1.5.5.1 The SP algorithm

Denote by $\boldsymbol{A}^\dagger = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$ the pseudoinverse of a matrix $\boldsymbol{A}$. Then, the OLS solution $\boldsymbol{y}$ to an overdetermined system $\boldsymbol{A}\boldsymbol{y} = \mathcal{Y}$ can be written as $\boldsymbol{y} = \boldsymbol{A}^\dagger\mathcal{Y}$.

**Initialization:**

- Given $K$, the desired number of nonzero coefficients ($2K \leq \min\{N, P\}$)
- Given the set of candidate regressors $\{\psi_{\boldsymbol{\alpha}} : \boldsymbol{\alpha} \in \mathcal{A}\}$ and the associated regression matrix $\boldsymbol{\Psi}$
- Initial active set: $\mathcal{A}^0 = \{K$ indices corresponding to the largest-in-magnitude entries in $\boldsymbol{\Psi}^T\mathcal{Y}\}$
- Denote by $\boldsymbol{\Psi}_{\mathcal{A}^0}$ the submatrix corresponding to $\mathcal{A}^0$
- Residual vector: $\boldsymbol{r}_0 = \mathcal{Y} - \boldsymbol{\Psi}_{\mathcal{A}^0}(\boldsymbol{\Psi}_{\mathcal{A}^0})^\dagger\mathcal{Y}$

**Iterative algorithm:** In step $j = 1, 2, \ldots$,

1. Define the temporary set
   $\mathcal{S} = \mathcal{A}^{j-1} \cup \{K$ indices corresponding to the largest-in-magnitude entries in $\boldsymbol{\Psi}^T\boldsymbol{r}_{j-1}\}$
   containing $2K$ regressors.
2. Compute the corresponding coefficients by OLS: $\boldsymbol{y} = (\boldsymbol{\Psi}_{\mathcal{S}})^\dagger\mathcal{Y}$.
3. Update the active set
   $\mathcal{A}^j = \{K$ indices corresponding to the largest-in-magnitude entries in $\boldsymbol{y}\}$.
4. Update residual: $\boldsymbol{r}_j = \mathcal{Y} - \boldsymbol{\Psi}_{\mathcal{A}^j}(\boldsymbol{\Psi}_{\mathcal{A}^j})^\dagger\mathcal{Y}$.
5. If $\|\boldsymbol{r}_j\|_2 \geq \|\boldsymbol{r}_{j-1}\|_2$, stop the iteration and return $\mathcal{A}^{j-1}$ and the associated coefficient vector and leave-one-out error. Otherwise, set $j = j + 1$ and continue with step 1.

The hyperparameter $K$ of SP can be determined, *e.g.*, by leave-one-out cross-validation (LOO). In this case, the above algorithm is run for a range of reasonable values for $K$. The final solution is the one with the lowest LOO error. Note that the LOO error can be computed inexpensively as in Eq. (1.20), since SP uses OLS on each subset of regressors.

### 1.5.6 Sparse PCE: Bayesian compressive sensing

A wholly different approach to the sparse regression problem is *Bayesian compressive sensing* (BCS), where the regression problem is rewritten in a Bayesian framework. Likelihood, priors, and auxiliary parameters are chosen to encourage sparsity in the regression coefficients. The coefficients are computed to be the maximum-a-posteriori (MAP) estimate given the model evaluations.
One of the first publications on BCS was the relevance vector machine (RVM) by Tipping (2001). Sargsyan et al. (2014) proposed the use of BCS in the context of PCE. Here, we use the BCS formulation of Babacan et al. (2010) (Fast Laplace), which is a generalization of

RVM attaining a provably sparser solutions. An illustration of the BCS framework is shown in Fig. 4.

Each of the quantities and auxiliary variables is considered to be random with a certain parametrized distribution, except for $\sigma^2$, which is chosen beforehand (see below), and $\nu = 0$. More precisely, the assumptions on parameters and distributions are as follows. The model evaluations $\mathcal{Y}$ are considered as i.i.d. realizations of a random variable parametrized by coefficients $\boldsymbol{y}$ and a noise variance parameter $\sigma^2$: $p(\mathcal{Y}|\boldsymbol{y}, \sigma^2) = \mathcal{N}(\mathcal{Y}|\boldsymbol{\Psi}\boldsymbol{y}, \sigma^2 \mathbf{1}_N)$ (likelihood). The noise variance is a given (fixed) hyperparameter of the algorithm, while the coefficients $\boldsymbol{y}$ are again random variables, each normally distributed with its own variance $\gamma_i$: $p(y_i|\gamma_i) = \mathcal{N}(y_i|0, \gamma_i)$. The $\gamma_i$'s are i.i.d. following an exponential distribution with shared parameter $\lambda$: $p(\gamma_i|\lambda) = \text{Exp}(\gamma_i|\frac{\lambda}{2})$. Finally, $\lambda$ is the last layer of the hierarchical framework, drawn from a Gamma-distribution $p(\lambda|\nu) = \Gamma(\lambda|\frac{\nu}{2}, \frac{\nu}{2})$ with $\nu \to 0$ (resulting in an uninformative, improper prior $p(\lambda) \propto \frac{1}{|\lambda|}$).

The goal of the algorithm is to compute the posterior distribution $p(\boldsymbol{y}, \boldsymbol{\gamma}, \lambda|\mathcal{Y})$ for a set of given model evaluations $\mathcal{Y}$. From this, the MAP estimate for $\boldsymbol{y}$ can be determined. However, the prior distributions of BCS are chosen to encourage sparsity, but are not conjugate, so that an analytical solution of the problem is not feasible. Instead, a fast approximate algorithm is employed, which iteratively updates the different auxiliary quantities as well as the coefficient vector estimate $\boldsymbol{y}$.

It has been found that BCS performs especially well for high-dimensional problems and in cases where only a small set of data is available (Lüthen et al., 2020).

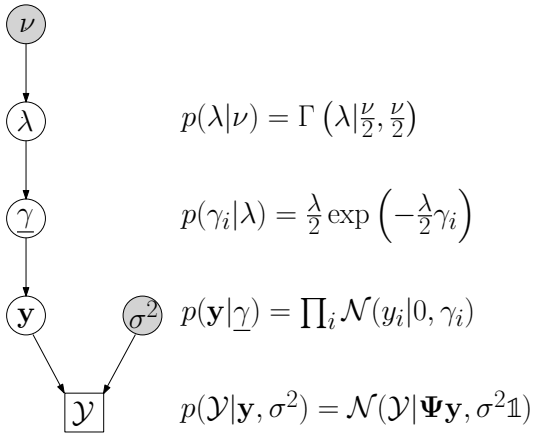



Figure 4: Illustration of the BCS setup by Babacan et al. (2010). All quantities are random variables with distributions parametrized by hyperparameters. The hierarchical structure and the choice of priors and hyperpriors results in a sparsity-encouraging posterior distribution for the coefficients $\boldsymbol{y}$.

#### 1.5.6.1 The BCS algorithm

The idea of the BCS algorithm as proposed by Babacan et al. (2010) (called Fast Laplace) is to maintain a vector $\boldsymbol{\gamma}$ of coefficient variances. If $\gamma_i = 0$, the associated coefficient $y_i$ must be zero as well, and the corresponding term in the basis is inactive. Else, if $\gamma_i > 0$, the term is

active. In each iteration, one regressor is chosen, and to be either added to the basis ($\gamma_i$ is set to a value $> 0$), deleted from the basis ($\gamma_i = 0$), or reassessed by means of a re-estimation of its variance $\gamma_i$.
The objective function $\mathcal{L}$ is the logarithm of $p(\boldsymbol{\gamma}, \lambda, \mathcal{Y})$, which has an analytical expression and can be differentiated to obtain the update equations. We refer to Babacan et al. (2010) and (Tipping and Faul, 2003, Appendix A) for further details.

**Initialization:**

- Given model evaluations $\mathcal{Y}$ and the corresponding matrix of regressor evaluations $\boldsymbol{\Psi}$
- Given the noise variance $\sigma^2$
- Given stopping threshold $\eta$
- Add the single regressor that has the largest correlation with the data to the model

**Iterative algorithm:**

1. For each of the regressors, compute the hypothetical updated value of $\gamma_i$ if this regressor alone was updated, and the associated hypothetical change in $\mathcal{L}(\boldsymbol{\gamma})$. Choose the regressor $i$ that accounts for the largest increase in $\mathcal{L}$.

2. If none of the regressors results in an increase in $\mathcal{L}$, or if the increase in $\mathcal{L}$ divided by the current value of $\mathcal{L}$ has been smaller than $\eta$ twice in a row, stop.

3. Add/remove/re-estimate: depending on the old and the new value of $\gamma_i$, the change corresponds to either removal, addition, or re-estimation of this regressor. All other quantities have to be updated accordingly. Continue with step 1.

The hyperparameter $\sigma^2$ can be chosen by $k$-fold cross-validation as follows. A number of candidate values for $\sigma^2$ is generated. The available data $\mathcal{Y}$ is divided into $k$ equally sized chunks. Each chunk is in turn treated as validation set, while the remaining data is used to train the model. Averaging over the $k$ validation errors gives the corresponding cross-validation error. This is done for each value of $\sigma^2$. The $\sigma^2$ with the smallest cross-validation error is chosen. Finally, BCS is run one more time with the chosen value of $\sigma^2$ and the full experimental design to get the final solution.
Throughout this document, algorithms like degree-adaptivity (Section 1.3.4) are described using the LOO error $\epsilon_{LOO}$, but hold equivalently for BCS by replacing the LOO error with the $k$-fold CV error $\epsilon_{CV}$.

## 1.6 Post-processing

The polynomial chaos expansion can be post-processed to obtain further information about the quantities of interest.

### 1.6.1 Moments of a PCE

Due to the orthonormality of the polynomial basis, the first two moments of a PCE are encoded in its coefficients. In particular, the mean value of a PCE reads:

$$\mu^{PC} = \mathbb{E}\left[\mathcal{M}^{PC}(\boldsymbol{X})\right] = y_{\mathbf{0}} \tag{1.37}$$

where $y_{\mathbf{0}}$ is the coefficient of the constant basis term $\Psi_{\mathbf{0}} = 1$.
Similarly, the variance of a PCE reads:

$$(\sigma^{PC})^2 = \mathbb{E}\left[(\mathcal{M}^{PC}(\boldsymbol{X}) - \mu^{PC})^2\right] = \sum_{\substack{\boldsymbol{\alpha}\in\mathcal{A} \\ \boldsymbol{\alpha}\neq\mathbf{0}}} y_{\boldsymbol{\alpha}}^2 \tag{1.38}$$

where the summation is over the coefficients of the non-constant basis elements only.

### 1.6.2 Uncertainty propagation and sensitivity analysis

When the polynomial coefficients are known, it is straightforward to evaluate the metamodel on new samples of the input random vector $\boldsymbol{X}$. In fact, it is sufficient to directly apply Eq. (1.2) by first evaluating the multivariate polynomials on the new sample and summing them weighted by their coefficients. The computational costs to perform this operation are limited to the evaluation of the univariate polynomials on the new sample and a small number of matrix multiplications, hence making this operation very efficient. This property can be used effectively for calculating the PDF of the model response accurately by using large Monte-Carlo samples of the inputs and, e.g., Kernel smoothing techniques.
Another important property of PCE is that the coefficients encode important information about the ANOVA decomposition of the surrogate model, which can be exploited to effectively calculate global sensitivity indices at very limited costs. The reader is referred to the UQLAB User Manual – Sensitivity Analysis module for further information on the relation between global sensitivity analysis and polynomial chaos expansions.

## 1.7 Bootstrap-based error estimates

The PCE module of UQLAB offers the possibility to estimate the accuracy of a local prediction through bootstrap resampling.

### 1.7.1 Bootstrap PCE

Marelli and Sudret (2018) have shown how the bootstrap resampling technique (Efron, 1979) can be applied to PCE to provide a local error estimator, calling this method *bPCE*. Resampling with substitution is used on the experimental design $\boldsymbol{\mathcal{X}}$, thus generating a set of bootstrap replications. Each replication contains the same number of sample points as the original experimental design $\boldsymbol{\mathcal{X}}$. Each of the $B$ replications $\{\boldsymbol{\mathcal{X}}^{(1)}, \ldots, \boldsymbol{\mathcal{X}}^{(B)}\}$ is used to calculate a corresponding PCE. This yields a set of $B$ different PCEs with coefficients

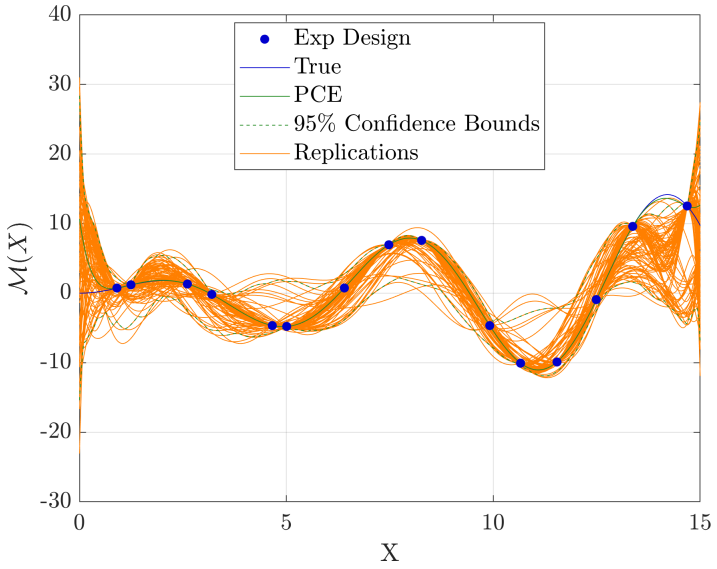



Figure 5: Function $y = x\sin(x)$ approximated by a PCE with confidence bounds from bootstrapping.

$\boldsymbol{y}_{\boldsymbol{\alpha}}^{(b)}$, $b = 1, \ldots, B$ and, consequently, a set of $B$ responses at each prediction point $\boldsymbol{x}$. Such sets of responses can be used to calculate the local variance (or other statistics of interest such as quantiles) of the PCE predictor, due to the finite size of the experimental design. An example of bPCE on a 1-dimensional function is illustrated in Figure 5, where the replications are represented as thin orange lines, and $95\%$ confidence bounds are derived from empirical quantiles on the bootstrap samples.

### 1.7.2 Fast bPCE

In practice, setting up a PCE for each bootstrap sample can be time consuming, especially when using sparse expansion techniques in high dimension or when an already large experimental design is available. Since for most applications local error estimates do not need to be very accurate, *fast bPCE* is used instead in UQLAB. In this approach, a sparse PCE basis is established only once using a sparse regression algorithm (*e.g.*, LAR or OMP) on the original experimental design $\boldsymbol{\mathcal{X}}$. The coefficients $\boldsymbol{y}_{\boldsymbol{\alpha}}^{(b)}$, $b = 1, \ldots, B$ are then calculated by OLS regression for each bootstrap sample.

# Chapter 2

# Usage

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

## 2.1 Reference problem: the Ishigami function

Polynomial chaos expansion aims at approximating a computational model with a polynomial surrogate. To this end, we will make use of a well-known benchmark for polynomial chaos expansions: the Ishigami function (Ishigami and Homma (1990), `www.sfu.ca/~ssurjano/ishigami.html`). It is an analytical 3-dimensional function characterized by non-monotonicity and high non-linearity, given by the following equation:

$$f(\boldsymbol{x}) = \sin(x_1) + a\sin^2(x_2) + bx_3^4\sin(x_1) \tag{2.1}$$

where the parameters are set to $a = 7$ and $b = 0.07$ in this example (see *e.g.* Ishigami and Homma (1990)).
The input random vector consists of three *i.i.d.* uniform random variables $X_i \sim \mathcal{U}(-\pi, \pi)$. An example UQLAB script that showcases several of the currently available PCE expansion techniques on the Ishigami function can be found in the example file:
`Examples/PCE/uq_Example_PCE_01_Coefficients.m`

## 2.2 Problem set-up

Recalling Eq. (1.3), truncated PCE reads:

$$\mathcal{M}(\boldsymbol{X}) \approx \sum_{\boldsymbol{\alpha}\in\mathcal{A}} y_{\boldsymbol{\alpha}}\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X})$$

The main ingredients that need to be set-up in a PCE analysis are:

- A model to surrogate $Y = \mathcal{M}(\boldsymbol{X})$;
- A probabilistic input model (random input vector $\boldsymbol{X}$);

- A truncated polynomial basis defined by $\mathcal{A}$;
- The polynomial coefficients $\{y_{\boldsymbol{\alpha}},\ \ \boldsymbol{\alpha} \in \mathcal{A}\}$.

### 2.2.1 Full model and probabilistic input model

The model in Eq. (2.1) is implemented as a Matlab `m-file` in:

`Examples/SimpleTestFunctions/uq_ishigami.m`

To surrogate it using UQLAB, we need to first configure a basic MODEL object:

```
MOpts.mFile = 'uq_ishigami' ;
myModel = uq_createModel(MOpts);
```

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

The three independent input variables can be defined as:

```
for ii = 1 : 3
  IOpts.Marginals(ii).Type = 'Uniform' ;
  IOpts.Marginals(ii).Parameters = [-pi, pi] ;
end
myInput = uq_createInput(IOpts);
```

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

## 2.3 Set-up of the polynomial chaos expansion

The PCE module creates a MODEL object that can be used as any other model. Its configuration options, however, are generally more complex than for a basic full model definition like the one in Section 2.2.1.

The basic options common to any PCE metamodelling MODEL read:

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

The input dimension of the problem $M$ is automatically retrieved by the configuration of the INPUT module given above. The additional configuration options needed to properly create a PCE object in UQLAB are given in the following sections.

## 2.4 Orthogonal polynomial basis

### 2.4.1 Univariate polynomial types

For most practical applications, it is not necessary in UQLAB to manually specify the univariate polynomial types that form the multivariate polynomial basis $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X})$ via Eq. (1.5). The default behaviour of UQLAB is to choose univariate polynomials depending on the distributions of the input variables, according to Table 2. Internally, a linear isoprobabilistic

transform (see Section 1.3.2.1) to a distribution from the same family with standard parameters is automatically performed (*e.g.*, $\mathcal{U}(a,b)$ is transformed to $\mathcal{U}(0,1)$). For non-classical input distributions, the univariate orthonormal polynomials are computed numerically by applying the Stieltjes procedure (see Section 1.3.2.2).

Table 2: 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 |
| $\Gamma(\lambda,\kappa)$ | Laguerre$(\lambda,\kappa)$ |
| $\mathcal{B}(r,s,a,b)$ | Jacobi$(r,s)$ |
| $\log\mathcal{N}(\mu,\sigma)$ | Hermite |
| Other | Arbitrary |

Note that the Jacobi and Laguerre polynomials are defined parametrically with the Beta and Gamma distribution parameters respectively. When a distribution does not belong to the above, the recurrence terms will be numerically computed if the integral of the distribution itself can be estimated accurately with numerical integration. Otherwise the Hermite polynomials will be used for distributions that do not have bounded support and Legendre polynomials when distributions have bounded support.
It is possible, however, to manually force the univariate polynomial families to the desired value by specifying the `PolyTypes` option in the input. As an example, to force the use of Hermite polynomials in the first dimension, Legendre polynomials in the second, and numerically compute the orthonormal polynomials for the third direction one has to specify:

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

In case of constant input variables, the corresponding `PolyTypes` entry would be `'Zero'`.
In case Laguerre or Jacobi polynomials are selected with `PolyTypes`, it is necessary for the `PolyTypesParams` option to be defined. The definition of the parameters of the polynomial families are consistent with that of their respective distributions and the redundant parameters, such as the bounds of the beta distribution for Jacobi polynomials, are ignored. For example one can specify:

```
MetaOpts.PolyTypes       = {'Hermite', 'Jacobi'     , 'Laguerre'};
MetaOpts.PolyTypesParams = {[]       , [2, 3, 0, 1], [3, 4]     };
```

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

### 2.4.2 Truncation schemes

The default truncation strategy in UQLab is the standard total-degree truncation scheme in Eq. (1.12), with maximum degree $p = 3$. To specify a desired maximum polynomial degree it is sufficient to add the `Degree` field to the `MetaOpts` configuration variable. To specify a maximum polynomial degree of *e.g.* 10 one can add:

```
MetaOpts.Degree = 10;
```

#### 2.4.2.1 Basis truncation

Additionally, one can configure any of the truncation strategies described in Section 1.3.3.1 and 1.3.3.2 with the optional `TruncOptions` field. A $q$-norm truncation with $q = 0.75$ and maximum rank $r = 2$ can be specified as follows:

```
MetaOpts.TruncOptions.qNorm = 0.75;
MetaOpts.TruncOptions.MaxInteraction = 2;
```

The two truncation schemes are not mutually exclusive and they can be specified either one-at-a-time or both together.

#### 2.4.2.2 User-specified basis

It is also possible to directly specify the set of multi-indices $\mathcal{A}$ that will be used to generate the multivariate polynomial basis. This can be accomplished by manually specifying the $P \times M$ matrix of polynomial degrees in the `TruncOptions.Custom` variable. As an example, one can specify a basis with $M = 3$, $p = 2$ and basis elements $\Psi_{0,0,0}(\boldsymbol{x}), \Psi_{0,2,0}(\boldsymbol{x}), \Psi_{1,0,0}(\boldsymbol{x})$ and $\Psi_{0,1,1}(\boldsymbol{x})$, as follows:

```
MetaOpts.TruncOptions.Custom = [0 0 0; 0 2 0; 1 0 0; 0 1 1];
```

In this case, it is not necessary to specify the degree of the basis. It is computed automatically from the user-specified basis.

## 2.5 Experimental design

For projection-based PCE, the experimental design is determined by the choice of quadrature scheme and degree (see Section 2.6.1).

For regression methods, there is more freedom in the choice of experimental design. At least the number of experimental design points has to be specified:

```
MetaOpts.ExpDesign.NSamples = 1000;
```

By default, samples are generated by Latin Hypercube sampling (LHS).
Several other options are available for the creation of the experimental design. A summary of the most common is given in the following.

- **Specify a sampling strategy:** it is possible to specify another sampling strategy by adding a `ExpDesign.Sampling` option. The following specifies sampling from a Sobol' pseudorandom sequence:

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

- **Manually specify an experimental design:** it is common to create PCE from already existing data. There are two ways to import data in a UQLAB PCE MODEL:

  – Specify the values of `ExpDesign.X` and `ExpDesign.Y` directly. Assuming the existing data is stored in the local variables `X_ED` and `Y_ED`, where each row of these variables corresponds to one point from the experimental design, and `X_ED` has as many columns as there are input random variables, 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.

**Note:** When an experimental design is specified manually, there is no need to create a MODEL object as in Section 2.2. However, an INPUT module with an input random vector compatible with the provided experimental design *must* be defined. This is an intrinsic property of PCE: $f_{\boldsymbol{X}}$ is needed to calculate the PCE coefficients.

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

## 2.6 Calculation of the coefficients

The remaining ingredient needed to complete the PCE is the set of polynomial coefficients $y_{\boldsymbol{\alpha}}$. In this section, the techniques introduced in Section 1.5 are deployed in UQLAB.

### 2.6.1 Projection: Gaussian quadrature

Calculating the PCE coefficients with Gaussian quadrature does not require any special configuration. Due to the very high non-linearity of the Ishigami function, a relatively high polynomial degree of $p = 14$ is needed to achieve satisfactory accuracy.
A projection-based PCE can be created with the following lines of code (note that for quadrature-based calculation of the coefficients no truncation scheme is necessary, as all the coefficients up to the specified degree are calculated simultaneously anyway):

```
% Reporting the previous configuration options as a reminder
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'PCE';

% Specification of 14th degree, Gaussian quadrature-based projection
```

```
MetaOpts.Degree = 14;
MetaOpts.Method = 'quadrature';

% Creation of the metamodel:
myPCE_Quadrature = uq_createModel(MetaOpts);
```

Once the model is created, a report with basic information about the PCE can be printed out as follows:

```
uq_print(myPCE_Quadrature)
```

which produces the following output:

```
%------------ Polynomial chaos output ------------%
Number of input variables:    3
Maximal degree:               14
q-norm:                       1.00
Size of full basis:           680
Size of sparse basis:         680
Full model evaluations:    3375
Quadrature error:          3.8654334e-13
Mean value:                    3.5000
Standard deviation:            3.7208
Coef. of variation:          106.309%
%-------------------------------------------------%
```

Similarly, a visual representation of the spectrum of the resulting non-zero coefficients can be visualized graphically as follows:

```
uq_display(myPCE_Quadrature);
```

which produces the image in Figure 6.

#### 2.6.1.1 Accessing the results

**Coefficients and basis**

All the information needed to evaluate Eq. (1.3) are available in the output structure `myPCE_Quadrature.PCE`:

```
myPCE_Quadrature.PCE
ans =
  Basis: [1x1 struct]
  Coefficients: [680x1 double]
  Moments: [1x1 struct]
```

The `PCE.Coefficients` array contains the coefficients in column vector format for all of the 680 $\boldsymbol{\Psi}_{\boldsymbol{\alpha}}(\boldsymbol{X})$ basis elements. The corresponding basis elements are given by the `PCE.Basis` structure:

```
myPCE_Quadrature.PCE.Basis
```

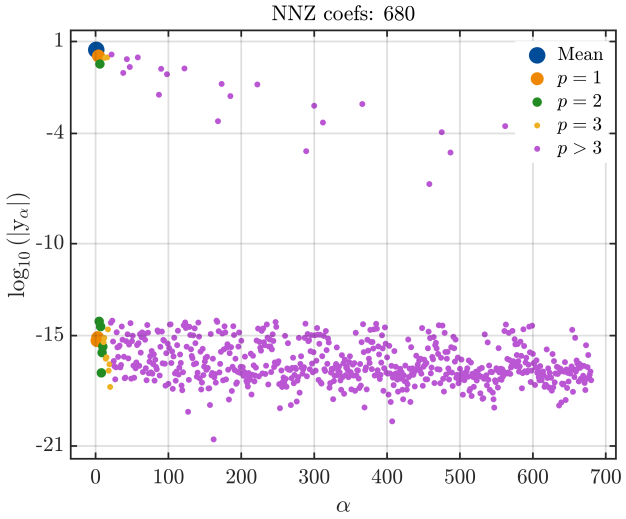



Figure 6: Graphical representation of the logarithmic spectrum of the PCE coefficients computed by quadrature. Most of the coefficients of the 680 basis elements are close to 0.

```
ans =
  PolyTypes: {3x1 cell}
  Indices: [680x3 double]
  MaxCompDeg: [14 14 14]
  MaxInteractions: 3
  Degree: 14
```

The `Basis.PolyTypes` cell array contains the $M$ univariate polynomial families $\phi^{(i)}$ in Eq. (1.5). The `Basis.Indices` matrix contains the $\boldsymbol{\alpha}$ multi-indices in Eq. (1.3) in row-vector format (in other words, the index set in $\mathcal{A}^{M,p,q}$ set in Eq. (1.14)). To each row of `Basis.Indices` corresponds a coefficient in the array `PCE.Coefficients`. The additional fields contain respectively:

- `Basis.MaxCompDeg`: the maximum univariate polynomial degree for each input variable for the basis elements with non-zero coefficients.
- `Basis.MaxInteractions`: the maximum rank of the basis elements with non-zero coefficients.
- `Basis.Degree`: the maximum degree of the basis elements with non-zero coefficients.

Finally, the `PCE.Moments` structure contains mean and variance of the model as calculated from the PCE (Eqs. (1.37),(1.38)).

For more details about the available information in the PCE output, refer to Section 3.2.1.

#### Model evaluations

The quadrature points and their corresponding model evaluations are stored in the structure `myPCE_Quadrature.ExpDesign`:

```
myPCE_Quadrature.ExpDesign
ans =
  Sampling: 'Quadrature'
  NSamples: 3375
  X: [3375x3 double]
  U: [3375x3 double]
  W: [3375x1 double]
  Y: [3375x1 double]
```

The `ExpDesign.Sampling` field contains information about the generation of the sample of model evaluations. In the case of quadrature-based projection method, it can only have the `'Quadrature'` value. The `ExpDesign.NSamples` field contains the total number of full-model evaluations that were run during the calculation.

The remaining fields contain, respectively:

- `ExpDesign.X`: the quadrature nodes where the model is evaluated;
- `ExpDesign.U`: the same points as `ExpDesign.X`, but rescaled and transformed onto the domain of definition of the orthogonal polynomials;
- `ExpDesign.Y`: the full model evaluation at each of the quadrature notes;
- `ExpDesign.W`: the quadrature weight of each point as in Eq. (1.26).

#### *A posteriori* error estimates

The structure `myPCE_Quadrature.Error` contains the normalized quadrature error estimate from equation Eq. (1.28).

### 2.6.1.2 Advanced options

There are several advanced options for the calculation of PCE coefficients with the projection method, namely selecting the Smolyak' sparse quadrature method in Section 1.5.1 and specifying the quadrature level.

- **Smolyak' scheme:** the Smolyak' quadrature scheme can be enabled by adding the following option:

  ```
  MetaOpts.Quadrature.Type = 'Smolyak';
  ```

  Note that up to dimension $M = 4$ Smolyak' scheme requires more nodes than full quadrature for the same level of accuracy.

- **Quadrature level:** the quadrature level (by default set to $l = p + 1$, where $p$ is the maximum polynomial degree) can be set to the desired value (*e.g.* $l = 15$) as:

```
MetaOpts.Quadrature.Level = 15;
```

For a comprehensive list of the options available for the quadrature method, see Table 5.

### 2.6.2 Ordinary Least-Squares (OLS)

The calculation of PCE coefficients with Ordinary Least-Squares on a sample of $N = 1,000$ model evaluations can be enabled with the following configuration:

```
% Reporting the previous configuration options as a reminder
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'PCE';

% Specification of 14th degree, OLS-based PCE
MetaOpts.Degree = 14;
MetaOpts.Method = 'OLS';

% Specification of the experimental design
MetaOpts.ExpDesign.NSamples = 1000;

% Creation of the metamodel:
myPCE_OLS = uq_createModel(MetaOpts);
```

Note that UQLAB will create the experimental design and evaluate the model response on it. Once the PCE is calculated, a report with basic information about the PCE results can be printed on screen by:

```
uq_print(myPCE_OLS);
```

which produces the following report:

```
%------------ Polynomial chaos output ------------%
Number of input variables:     3
Maximal degree:                14
q-norm:                        1.00
Size of full basis:            680
Size of sparse basis:          680
Full model evaluations:     1000
Leave-one-out error:           1.0283745e-08
Mean value:                     3.5000
Standard deviation:             3.7208
Coef. of variation:           106.309%
%-------------------------------------------------%
```

A visual representation of the spectrum of the resulting PCE coefficients can be created as follows:

```
uq_display(myPCE_OLS);
```

which produces the image in Figure 7.

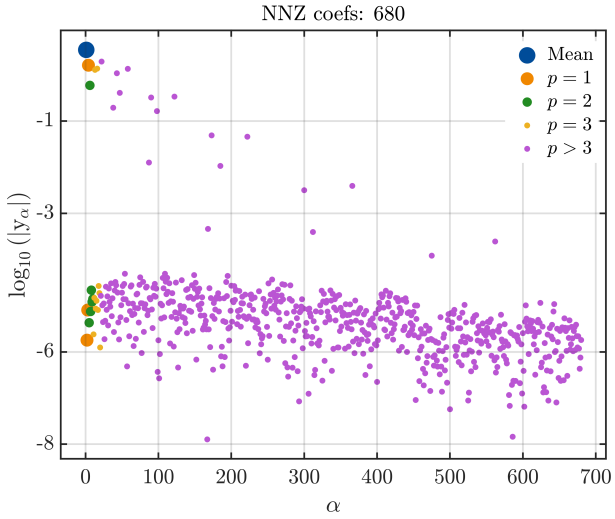



Figure 7: Graphical representation of the logarithmic spectrum of the PCE coefficients computed by OLS. Again, most of the coefficients of the 680 basis elements are close to 0.

#### 2.6.2.1 Accessing the results

**Coefficients and basis**

The coefficients and basis can be accessed from the structure `myPCE_OLS.PCE`:

```
myPCE_OLS.PCE
ans =
  Basis: [1x1 struct]
  Coefficients: [680x1 double]
  Moments: [1x1 struct]
```

**Model evaluations**

The model evaluations used to calculate the PCE coefficients with OLS can be accessed from the `myPCE_OLS.ExpDesign` structure:

```
myPCE_OLS.ExpDesign
ans =
  NSamples: 1000
  Sampling: 'LHS'
  X: [1000x3 double]
  U: [1000x3 double]
  Y: [1000x1 double]
```

The `ExpDesign.Sampling` field has the `'LHS'` value. It represents the sampling strategy used to create the experimental design, see Section 2.5 for advanced options for the creation of the experimental design.

***A posteriori* error estimates**

The *a posteriori* error estimates are stored in the `myPCE_OLS.Error` structure:

```
myPCE_OLS.Error
ans =
  LOO: 1.0284e-08
```

```
ModifiedLOO: 5.4698e-05
normEmpErr: 1.2706e-12
```

where `Error.normEmpErr` and `Error.LOO` corresponds to the empirical error estimate in Eq. (1.18) and to the modified leave-one-out error in Eq. (1.20), respectively.
Note that the `Error.normEmpErr` is much smaller than `Error.LOO`, as it does not account for over-fitting.
For a comprehensive overview of the outputs available for the OLS method, see Table 23.

#### 2.6.2.2 Advanced options

There are no OLS specific options *per se*. However, when combined with *degree adaptive PCE* described in Section 1.3.4 and 2.7.1, two parameters can be set to tune the convergence of the algorithm:

- **Target accuracy:** by default set to 0, it corresponds to $\epsilon_T$ in Section 1.3.4. It can be manually set to any value (*e.g.* $10^{-8}$) as follows:

```
MetaOpts.OLS.TargetAccuracy = 1e-8;
```

- **Disable the modified LOO estimator:** by default, $\epsilon_{LOO}$ is calculated with the modified estimator in Eq. (1.22) and the correction factor in (1.23). It is possible, however, to enable the classical LOO estimator in Eq. (1.19) as follows:

```
MetaOpts.OLS.ModifiedLOO = 0;
```

Additional configuration options are available for the creation of the experimental design from which the least-square regression is performed. They are listed in Section 2.5. A detailed list of the available configuration options for OLS can be found in Table 6.

### 2.6.3 Sparse regression methods (LARS, OMP, SP, BCS)

Configuring a sparse PCE with a sparse regression solver is very similar to setting up a PCE with OLS. All four sparse regression solvers available in UQLab (LARS, OMP, SP, BCS) are used in basically the same way.
The code needed to create a basic LARS-based PCE with 1,000 sample points in the experimental design is as follows:

```
% Reporting the previous configuration options as a reminder
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'PCE';

% Specification of 14th degree LARS-based PCE
MetaOpts.Degree = 14;
MetaOpts.Method = 'LARS';

% Specification of the experimental design
MetaOpts.ExpDesign.NSamples = 1000;
```

```
% Creation of the metamodel:
myPCE_sparse = uq_createModel(MetaOpts);
```

Instead of LARS, we could use OMP, SP, or BCS by assigning `'OMP'`, `'SP'`, or `'BCS'` to the field `MetaOpts.Method`.

A report with basic information about the PCE results can be printed on screen by:

```
uq_print(myPCE_sparse);
```

which produces the following report:

```
%------------ Polynomial chaos output ------------%
   Number of input variables:    3
   Maximal degree:               14
   q-norm:                       1.00
   Size of full basis:           680
   Size of sparse basis:         33
   Full model evaluations:       1000
   Leave-one-out error:          9.1109258e-12
   Modified leave-one-out error: 9.7524461e-12
   Mean value:                   3.5000
   Standard deviation:           3.7208
   Coef. of variation:           106.309%
%-------------------------------------------------%
```

A visual representation of the spectrum of the non-zero coefficients can visualized graphically as follows:

```
uq_display(myPCE_sparse)
```

which produces the image in Figure 8a. Notice how the LARS solution only produces 33 non-zero coefficients, which is much sparser than the OLS solution with its 680 non-zero coefficients (Figure 7). Similar observations can be made for the other three sparse solvers OMP (Figure 8b), SP (Figure 8c), and BCS (Figure 8d).

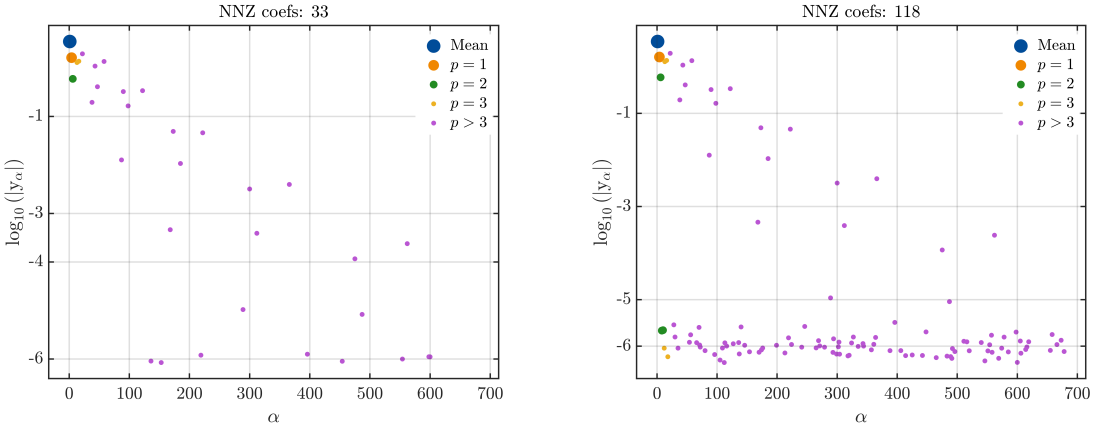
#### 2.6.3.1 Accessing the results

**Coefficients and basis**

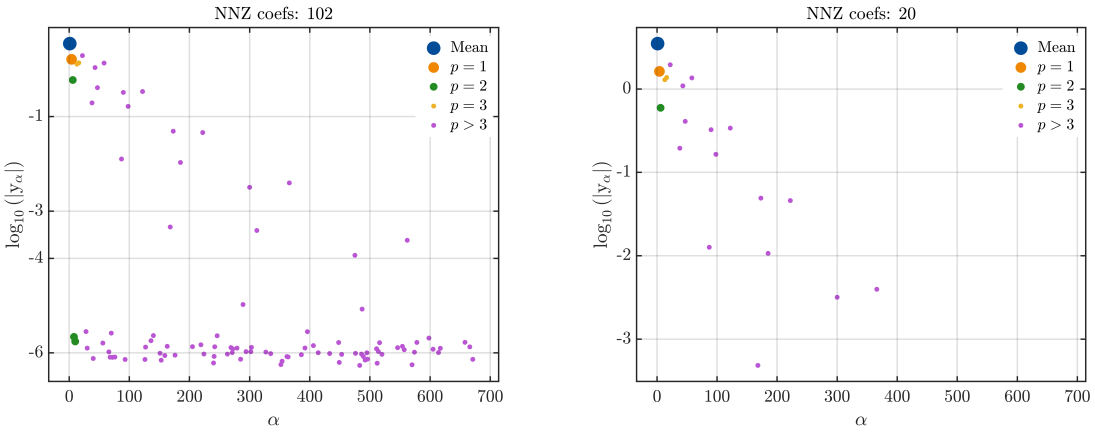
The coefficients and basis can be accessed from the structure `myPCE_sparse.PCE`:

```
myPCE_sparse.PCE
ans =
Basis: [1x1 struct]
Coefficients: [680x1 double]
Moments: [1x1 struct]
```

**Model evaluations**




(a) LARS (b) OMP




(c) SP (d) BCS

Figure 8: Graphical representation of the logarithmic spectra of the PCE coefficients computed by the sparse regression methods LARS, OMP, SP, and BCS. The sparsity of these solutions compared to the OLS solution in Figure 7 is clear.

The experimental design structure containing the model evaluations used to calculate the PCE coefficients is identical to that in OLS in Section 2.6.1.1.

```
myPCE_sparse.ExpDesign
ans =
  NSamples: 1000
  Sampling: 'LHS'
  X: [1000x3 double]
  U: [1000x3 double]
  Y: [1000x1 double]
```

The `ExpDesign.Sampling` field has the `'LHS'` value. It represents the sampling strategy used to create the experimental design. See Section 2.5 for advanced options for the creation of the experimental design.

### *A posteriori* error estimates

The *a posteriori* error estimates are stored in the `myPCE_sparse.Error` structure in the same

format as in Section 2.6.2.1:

```
myPCE_sparse.Error
ans =
  LOO: 9.1109e-12
  ModifiedLOO: 9.7524e-12
  normEmpErr: 8.3448e-12
```

Note that the LOO error for sparse PCE ($\approx 10^{-11}$) is significantly smaller than for OLS ($\approx 10^{-8}$), even though the experimental design has the same size $N = 1,000$.
Note that BCS relies on the $k$-fold cross-validation error instead of LOO. Therefore, in case of BCS the error reported in `myPCE_sparse.Error.LOO` (or `myPCE_sparse.Error.ModifiedLoo`) is not the LOO error, but the (modified) $k$-fold CV error.
For a comprehensive overview of further outputs available for the LARS method, see Table 24; for the OMP method, see Table 25.

#### 2.6.3.2 Shared advanced options

Albeit the default settings are optimal for most real case scenarios, the sparse solvers allow for the customization of several parameters that can be used to fine-tune the coefficients estimation.
For all three sparse regression method relying on the LOO error (*i.e.*, LARS, OMP, and SP), by default the modified LOO error estimate of Eq. (1.22) is used. For BCS, which relies on the $k$-fold CV error, the modification factor of Eq. (1.23) can be applied to the CV error, but by default it is not. This can be changed as follows:

- **Enable or disable the modified LOO (resp. CV) estimator:** by default, $\epsilon_{LOO}$ is calculated with the modified estimator in Eq. (1.22), using the correction factor in (1.23). It is possible, however, to enable the classical LOO (resp. CV) estimator in Eq. (1.19) as follows:

  ```
  MetaOpts.LARS.ModifiedLoo = 0;
  ```

  For OMP or SP, use the field `.OMP` or `.SP` instead of `.LARS`. For BCS, use the field `.BCS`, and note that while the option is called `ModifiedLoo`, it actually applies to the $k$-fold cross-validation error.

  Note that the classical estimator tends to be less sensitive to over-fitting, hence generally producing denser and less accurate PCE models.

Additional configuration options are available for the creation of the experimental design for which the sparse regression is performed. They are listed in Section 2.5.

#### 2.6.3.3 Advanced options for LARS

Advanced options for LARS can be specified with the `MetaOpts.LARS` structure as follows:

- **Disable the early stop criterion:** with some models, LARS can stop prematurely and yield inaccurate results. To disable the early stop criterion in Section 1.5.3.3, add:

```
MetaOpts.LARS.LarsEarlyStop = false;
```

> **Note:** Disabling this option can significantly increase the computational time necessary to calculate the coefficients. However, for small experimental designs (*i.e.* $N = 50$ points), it is disabled by default to improve accuracy.

- **Disable applying OLS for calculating LOO error:** by default, $\epsilon_{LOO}$ used for model selection during LARS iteration is calculated as in Eq. (1.20) (or Eq. (1.22)). As mentioned in Section 1.4.2, the use of Eq. (1.20) requires the PCE model $\mathcal{M}^{PC}(\boldsymbol{x})$ built with OLS. As a result, at the end of each LARS iteration step, a least-square minimization is applied to the selected basis functions. This OLS operation can be disabled by setting

```
MetaOpts.LARS.HybridLoo = 0;
```

  Nevertheless, Eq. (1.20) is still used for model selection. In this case, $\mathcal{M}^{PC}(\boldsymbol{x})$ in Eq. (1.20) corresponds to the PCE model whose coefficients are calculated by LARS update at each iteration.

- **Store the LARS iterations in memory:** by default UQLAB will not cache all the LARS iterations after the algorithm ends, because it may require significant memory resources. This behaviour can be changed as follows:

```
MetaOpts.LARS.KeepIterations = 1;
```

  If this option is active, a large array containing all of the coefficients for each LARS iteration is saved in: `myPCE_sparse.Internal.PCE.LARS.coeff_array`.

A detailed list of the available configuration options for LARS can be found in Table 7.

#### 2.6.3.4 Advanced options for OMP

Advanced options for OMP can be specified with the `MetaOpts.OMP` structure as follows:

- **Disable the early stop criterion:** with some models, OMP can stop prematurely and yield inaccurate results. To disable the early stop criterion in Section 1.5.4.2, add:

```
MetaOpts.OMP.OmpEarlyStop = false;
```

> **Note:** Disabling this option can significantly increase the computational time necessary to calculate the coefficients. However, for small experimental designs (*i.e.* $N = 50$ points), it is disabled by default to improve accuracy.

- **Store the OMP iterations in memory:** by default UQLAB will not cache all the OMP iterations after the algorithm ends, because it may require significant memory resources. This behaviour can be changed as follows:

```
MetaOpts.OMP.KeepIterations = 1;
```

  If this option is active, a large array containing all of the coefficients for each OMP iteration is saved in: `myPCE_OMP.Internal.PCE.OMP.coeff_array`.

A detailed list of the available configuration options for OMP can be found in Table 8.

#### 2.6.3.5 Advanced options for SP

SP has one hyperparameter, the number $K$ of nonzero coefficients. It can be specified explicitly by setting

```
MetaOpts.SP.NNZ = K;
```

If $K$ is not specified, it is determined by cross-validation from a range of 10 evenly spaced values between 1 and $\min\{\frac{N}{2}, \frac{P}{2}\}$ ($K = 1$ excluded). By default, this is done by LOO as described in 1.5.5.1. The user can also request $k$-fold cross-validation (default are $k = 5$ folds) by

```
MetaOpts.SP.CVMethod = 'kfold';
```

The number of folds can be set in the optional field `MetaOpts.SP.NumFolds`.
A detailed list of the available configuration options for SP can be found in Table 9.

#### 2.6.3.6 Advanced options for BCS

The hyperparameter $\sigma^2$ of BCS is determined by $k$-fold cross-validation (default: $k = 10$) from the following range of values: `sigma2_range = N * var(Y) * 10.^linspace(-16,-1,10)`.
The number of folds can be changed in the optional field `MetaOpts.BCS.NumFolds`.

### 2.6.4 Custom regression solvers

It is possible to connect custom sparse regression solvers to the UQLAB PCE module. To achieve this, one needs to write a wrapper code for the custom solver `mysolver`, `uq_PCE_mysolver` (note: only lowercase solver names are supported) that follows the input/output convention of the other solvers in UQLAB (see *e.g.* `uq_PCE_sp.m`). More details on the implementation are given further below.
When the wrapper is available, the custom solver can be used as follows:

```
MetaOpts.Method = 'MYSOLVER';
```

(note: this option is case-insensitive). All features of the PCE module, like basis truncation, basis adaptivity, custom experimental designs etc. can be used in the same way as for the other sparse solvers.

**Implementation details**

The wrapper function `uq_PCE_mysolver` has two arguments: the three-dimensional array `univ_p_val`, which contains the univariate polynomial evaluations on the experimental design, and the MODEL object `current_model`, which stores all information about the PCE that is being constructed. The return variable `results` must be a struct with the following fields:

- `.indices` : multi-indices defining the basis
- `.coefficients` : computed coefficients, in the same order as the multi-indices
- `.normEmpErr` : empirical error
- `.LOO` : leave-one-out error (or a similar criterion that can be used for model selection)
- `.optErrorParams`: a struct with two fields `.loo` and `.normEmpErr` (same values as above).
- If the modified LOO (Eq. 1.22) shall be used, this value must be computed inside `uq_PCE_mysolver` and assigned to `results.LOO` and `results.ModifiedLoo`. The unmodified LOO is stored in `results.optErrorParams.loo` as before.

## 2.7 Basis adaptivity

All regression methods (OLS, LARS, OMP, SP, and BCS) provide a cross-validation error estimator – the LOO error $\epsilon_{LOO}$ in case of OLS, LARS, OMP, and SP, and the $k$-fold cross-validation error $\epsilon_{CV}$ in case of BCS – which can be used to develop basis-adaptive PCE as described in Section 1.3.4.
In the following, everything described in terms of $\epsilon_{LOO}$ holds equivalently for $\epsilon_{CV}$ in case of BCS.

### 2.7.1 Degree-Adaptive PCE

Degree-adaptive PCE is automatically enabled in UQLAB if the `MetaOpts.Degree` option is an array of values instead of a single one. The following code creates a degree-adaptive sparse PCE (LARS) with $p \in [1, 30]$ from an experimental design with $N = 256$ for the Ishigami function:

```
% Reporting the previous configuration options as a reminder:
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'PCE';

% Specification of degree-adaptive LARS
MetaOpts.Degree = 1:30; % range of degrees to be tested
MetaOpts.Method = 'LARS';

% Specification of the experimental design
MetaOpts.ExpDesign.Sampling = 'Sobol';
MetaOpts.ExpDesign.NSamples = 256;
```

```
% Creation of the metamodel:
myPCE_LARSAdaptive = uq_createModel(MetaOpts);
```

Despite the smaller experimental design, the degree-adaptive PCE converges to a maximal PCE degree $p = 24$ with the lowest $\epsilon_{LOO}$ amongst the examples presented in this section:

```
myPCE_LARSAdaptive.Error
ans =
  LOO: 1.0844e-17
  ModifiedLOO: 2.6076e-17
  normEmpErr: 3.6210e-18
```

The resulting coefficients spectrum is shown in Figure 9.

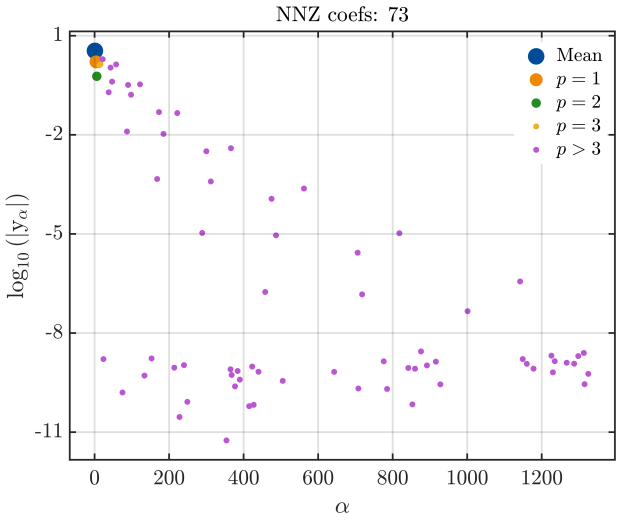



Figure 9: Graphical representation of the logarithmic spectrum of the PCE coefficients for degree-adaptive PCE. The analysis converged to $p = 24$ with $N = 256$.

It is therefore recommended to always specify in the PCE options a range of polynomial degrees when using least-square methods, so as to allow adaptive PCE to adaptively choose the best polynomial degree given the experimental design specifications.

#### 2.7.1.1 Accessing the results

The outputs of a degree-adaptive PCE analysis are unchanged from their non-adaptive counterparts, because only the iteration with the best $\epsilon_{LOO}$ is stored. The best degree is stored in `myPCE.PCE.Basis.Degree`.

#### 2.7.1.2 Advanced options

The default behaviour of the degree-adaptive scheme is to automatically stop increasing the maximum degree if the $\epsilon_{LOO}$ has not decreased for at least two subsequent iterations of the algorithm. Experience shows that once over-fitting is detected with $\epsilon_{LOO}$ on an experimental design, further increasing the size of the polynomial basis results in worse PCE models.

In some rare cases, however, the algorithm can stop prematurely due to a local minimum in the $\epsilon_{LOO}$ *vs.* $p$ curve. This can be prevented by setting the `MetaOpts.DegreeEarlyStop` flag to false:

```
MetaOpts.DegreeEarlyStop = false;
```

When disabled, all the degrees specified in the `MetaOpts.Degree` array will be calculated, and the best candidate will be chosen only at the end.

**Note:** Disabling this option can significantly increase the computational costs of the PCE coefficients calculation, as the size of the polynomial basis (and hence the number of coefficients that need to be calculated) increases very rapidly with the maximum polynomial degree.

### 2.7.2 q-norm-Adaptive PCE

Due to the sparsity-of-effects principle (see Section 1.5.3) it is advantageous to use a truncation scheme when searching for an optimal basis. To this end, UQLAB offers a q-norm-adaptive PCE set-up (for the q-norm truncation scheme see Section 1.3.3.2). The q-norm adaptivity also makes use of the $\epsilon_{LOO}$ error estimator and is automatically enabled in UQLAB if the `MetaOpts.TruncOptions.qNorm` option is an array of values instead of a single one. The q-norm adaptivity can be combined with the degree adaptivity. The following code creates a q-norm- and degree-adaptive sparse PCE (LARS) with q-norm $\in [0.5, 0.6, \ldots, 1]$ and $p \in [1, 30]$ from an experimental design with $N = 256$ for the Ishigami function:

```
% Reporting the previous configuration options as a reminder:
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'PCE';

% Specification of degree- and q-norm-adaptive LARS
MetaOpts.TruncOptions.qNorm = 0.5:0.1:1; % q-norms to be tested
MetaOpts.Degree = 1:30;
MetaOpts.Method = 'LARS';

% Specification of the experimental design
MetaOpts.ExpDesign.Sampling = 'Sobol';
MetaOpts.ExpDesign.NSamples = 256;

% Creation of the metamodel:
myPCE_LARSAdaptive = uq_createModel(MetaOpts);
```

**Note:** When setting up a low-dimensional q-norm-adaptive PCE, small q-norm increments will not affect the basis.

#### 2.7.2.1 Accessing the results

As in the case of degree-adaptive PCE, the outputs of a q-norm-adaptive PCE analysis are unchanged from their non-adaptive counterparts, because only the iteration with the best $\epsilon_{LOO}$ is stored. The best q-norm is stored in `myPCE.PCE.Basis.qNorm`.

#### 2.7.2.2 Advanced options

The q-norm-adaptive scheme starts for every polynomial degree with the smallest specified q-norm. The stopping criterion is a little more complex than the one for the degree-adaptive scheme. For small polynomial degrees a small increase in the q-norm might not allow for additional basis functions and will consequently not affect the set up PCE or its $\epsilon_{LOO}$. For this reason, UQLAB only regards iterations as interesting if they affect the $\epsilon_{LOO}$ or the basis size of the PCE. The scheme automatically stops increasing the q-norm once $\epsilon_{LOO}$ did not decrease in two subsequent interesting iterations. It also stops if the $\epsilon_{LOO}$ stays the same but the basis size increases twice.

The early stopping mechanism can be disabled by setting the `MetaOpts.qNormEarlyStop` flag to false:

```
MetaOpts.qNormEarlyStop = false;
```

When disabled, for each degree, all q-norms specified in the `MetaOpts.TruncOptions.qNorm` array will be calculated, and the best candidate will be chosen only at the end.

**Note:** As in the case of the degree-adaptive scheme, disabling this option can dramatically increase the computational costs of the PCE coefficients calculation. Especially for high degrees and q-norm arrays with many entries, the computational cost increases rapidly. However, *when the number of coefficients is kept small (*e.g. *less than* $50$ *points) the early stop mechanism should be turned off to ensure the best fit can be found*. In any case, when the experimental design is small, there are not many calculations to be done to construct the metamodel.

## 2.8 Use of a validation set

If a validation set is provided (see Table 12 in Section 3.1.10.), UQLAB automatically computes the validation error given in Eq. (1.17). 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 `myPCE.Error.Val` next to the other error measures (see Table 19 in Section 3.2.3) and will also be displayed when typing `uq_print(myPCE)`.

## 2.9 Manually specify inputs and computational models

The UQLAB framework allows one to create many INPUT and MODEL objects in the same session (see, e.g. UQLAB User Manual – the MODEL module and UQLAB User Manual – the INPUT module). The default behaviour of the PCE 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;
```

## 2.10 PCE of vector-valued models

The examples presented so far in this chapter dealt with scalar-valued models. In case the model (or the experimental design, if manually specified) has multi-component outputs (denoted by $N_{out}$), UQLAB performs an independent PCE for each output component on the shared experimental design. No additional configuration is needed to enable this behaviour. A PCE with multi-component outputs can be found in the UQLAB example in:
`Examples/PCE/uq_Example_PCE_04_MultipleOutputs.m`

### 2.10.1 Accessing the results

Running a PCE 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:

```
myPCE.PCE
ans =
1x9 struct array with fields:
  Basis
  Coefficients
  Moments
```

Each of elements of the `PCE` structure is functionally identical to its scalar counterpart in Section 2.6.1.1, 2.6.2.1 and 2.6.3.1.
Similarly, the `myPCE.Error` structure becomes a multi-element structure:

```
myPCE.Error
ans =
1x9 struct array with fields:
  LOO
  ModifiedLOO
  normEmpErr
```

## 2.11 Using a PCE as a predictor

Regardless on the strategy used to calculate the PCE coefficients or truncating the basis, the PCE model in Eq. (1.3) can be used to predict new points outside of the experimental design. Indeed, after a PCE 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).

### 2.11.1 Evaluate a PCE

Consider the Ishigami example in Section 2.1. After calculating the coefficients with any of the methods described in Section 2.6, one can evaluate the PCE metamodel on point $\boldsymbol{x} = \{0.3, -1.0, 2.2\}$ as follows:

```
X = [0.3 1.0 2.2];
% Evaluate the metamodel on the same input vector
YPC = uq_evalModel(X)
YPC =
5.9443
```

which can be compared to the true model:

```
YTrue = uq_ishigami(X)
YTrue =
5.9443
```

As most functions within UQLAB, model evaluations are vectorized, *i.e.* evaluating multiple points at a time is much faster than repeatedly evaluating one point at a time. To evaluate the response of the PCE metamodel on an input sample of size $N = 10^5$ in UQLAB, one can write (for details on how to use the input module to sample distributions, see the UQLAB User Manual – the INPUT module):

```
X = uq_getSample(1e5);
Y = uq_ishigami(X);
YPC = uq_evalModel(X);
```

The histogram and scatter plots of the `Y` and `YPC` vectors are given in Figure 10. Due to the high accuracy of the model, the original function and the metamodel are virtually indistinguishable.

### 2.11.2 Local error estimates

Getting local error estimates from a PCE consists of two parts: setting up a bootstrap PCE (bPCE; see Section 1.7.1) and using the bPCE to compute the local error estimates of the PCE point-wise predictions.

#### 2.11.2.1 Set up bPCE

To set up a bPCE, the number of bootstrap replications $B$ needs to be specified *before* creating the PCE metamodel as follows:

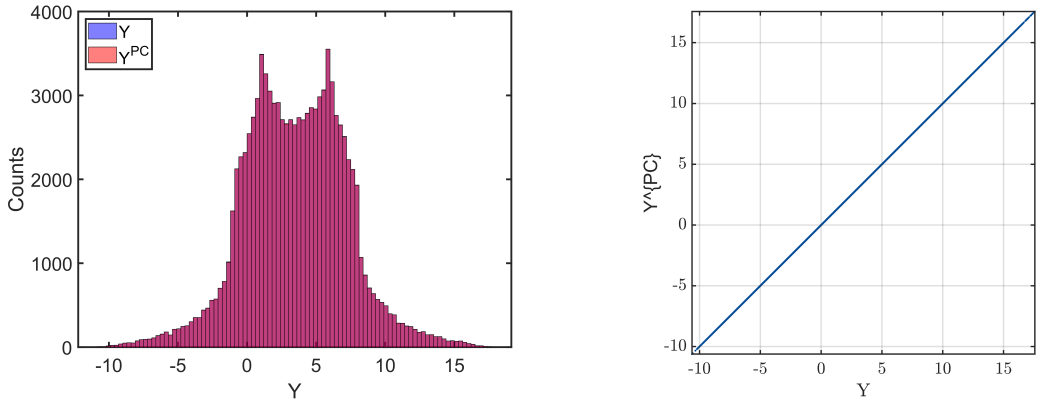



Figure 10: Histogram and scatter plots of true *vs.* metamodelled responses of the Ishigami function to a sample of the input of size $n = 10^5$.

```
MetaOpts.Bootstrap.Replications = 100;
```

The PCE produced by `uq_createModel` will be similar to the one set up without the bootstrap specification. The additional information is stored in `myPCE.Internal.Bootstrap` structure and used to compute the error estimates when evaluating the metamodel.

#### 2.11.2.2 Apply bPCE to get error estimates

After its creation, the bPCE can be applied on a sample using `uq_evalModel` to get predictions, just like a non-bootstrap PCE (see Section 2.11.1). The extended capabilities can be employed by calling more output arguments:

```
[YPC,YPC_var,YPC_replications] = uq_evalModel(myPCE,X);
```

Here, `YPC_var` provides the sample variance of the predictions at each point. `YPC_replications` is a $N \times B \times N_{out}$ matrix that contains the bPCE predictions for each of the $B$ bootstrap replications. Using the bPCE predictions and the MATLAB function `quantile` empirical quantiles can be employed to get error margins:

```
% 95% confidence interval for the prediction on the fifth point:
[loBound, upBound] = quantile(YYPC_BootSample(5,:),[0.025 0.975]);
```

## 2.12 Manually specifying PCE parameters (*predictor-only* mode)

It is also possible to use the PCE module in UQLAB to build custom PCE-based models that can be used as MODEL objects as in Section 2.11. This allows, e.g., to import a metamodel calculated with other software within the UQLAB framework, or even or to create one *ad-hoc*. In the following, we exemplify how to create a custom PCE with the following characteristics:

- standard normal input variables;
- up to second degree Hermite polynomials polynomial basis;

- only three non-zero coefficients: $y_{[0,0]} = 5$, $y_{[0,1]} = 1$, $y_{[1,1]} = 3$.

```
% 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 PCE
MetaOpts.Input = myInput;
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'PCE';
MetaOpts.Method = 'Custom';
% Basis: polynomial families
MetaOpts.PCE.Basis.PolyTypes = {'Hermite','Hermite'};
% Basis: polynomial alpha indices
MetaOpts.PCE.Basis.Indices = [0 0; 0 1; 1 1];
% PCE coefficients (same order as MetaOpts.PCE.Basis.Indices)
MetaOpts.PCE.Coefficients = [5; 1; 3];

% Create the metamodel
myPCE = uq_createModel(MetaOpts);

% 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.PCE(i)` structure.

## 2.13 Using a PCE with constant input variables

In some analyses, one may need to assign a constant value to one or to a set of input variables. When this is the case, the PCE metamodel is built by internally removing the constant variable 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 variables 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 input is set to zero, UQLAB automatically sets this variable'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 PCE metamodel

### Syntax

```
myPCE = uq_createModel(MetaOpts)
```

### Input

The struct variable `MetaOpts` contains the following fields:

Table 3: `MetaOpts`

| | | | |
|---|---|---|---|
| ● | `.Type` | `'Metamodel'` | Select the metamodelling tool |
| ● | `.MetaType` | `'PCE'` | Select polynomial chaos expansion |
| ☐ | `.Input` | INPUT object | Probabilistic input model (See Section 2.9) |
| ☐ | `.Name` | String | Unique identifier for the metamodel |
| ☐ | `.Display` | String<br>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. |
| ☐ | `.Degree` | Integer scalar | Maximum polynomial degree |
| | | Integer array | Set of polynomial degrees for degree-adaptive polynomial chaos (Section 2.7.1) |
| ☐ | `.DegreeEarlyStop` | Logical<br>default: true | Toggle polynomial degree early stop criterion on / off (Section 2.7.1). |
| ☐ | `.PolyTypes` | $1 \times M$ Cell array of strings | List of polynomial families to be used to build the PCE basis. The default choice is given in Table 2. 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 PCE basis. It is only used when `.PolyTypes` contains Jacobi or Laguerre polynomials. See Section 2.4 for usage example. |
| ☐ | `.TruncOptions` | Table 4 | Basis truncation (Section 1.3.3) |

| | | | |
|---|---|---|---|
| ☐ | .qNormEarlyStop | Logical<br>default: true | Toggle hyperbolic truncation norm early stop criterion on / off (Section 2.7.2). |
| ☐ | .Method | String<br>default: 'LARS' | Coefficients calculation method (Section 2.6) |
| | | 'Quadrature' | Quadrature (Section 1.5.1) |
| | | 'OLS' | Ordinary Least-Squares Regression (Section 1.5.2.1) |
| | | 'LARS' | Least-Angle Regression (Section 1.5.3) |
| | | 'OMP' | Orthogonal Matching Pursuit (Section 1.5.4) |
| | | 'SP' | Subspace Pursuit (Section 1.5.5) |
| | | 'BCS' | Bayesian Compressive Sensing (Section 1.5.6) |
| | | 'Custom' | User-defined PCE coefficients and basis (no calculations) |
| ⊞ | .Quadrature | Table 5 | Quadrature options (Section 2.6.1.2) |
| ⊞ | .OLS | Table 6 | OLS-specific options (Section 2.6.2.2) |
| ⊞ | .LARS | Table 7 | LARS-specific options (Section 2.6.3) |
| ⊞ | .OMP | Table 8 | OMP-specific options (Section 2.6.3) |
| ⊞ | .PCE | Table 15 | Custom-PCE parameters (Section 2.12). Use the same format as the default output of the calculation |
| ☐ | .FullModel | MODEL object | UQLab model used to create an experimental design (Section 2.9) |
| ☐ | .ExpDesign | Table 11 | Experimental design-specific options (Section 2.5) |
| ☐ | .ValidationSet | Table 12 | Validation set components (Section 2.8) |
| ☐ | .Bootstrap | Table 13 | Bootstrapping options (Section 2.11.2) |

### 3.1.1 Truncation options

The truncation strategies described in Section 1.3.3 can be specified with the TruncOptions field as described in Section 2.4.2. The full list of available options is reported in Table 4.

Table 4: MetaOpts.TruncOptions

| | | | |
|---|---|---|---|
| ☐ | `.qNorm` | Double<br>default: 1 | Hyperbolic truncation scheme (Section 1.3.3). Corresponds to $0 < q \leq 1$ in Eq. (1.14) |
| | | Double array | Set of hyperbolic norms to be tested for q-norm-adaptive polynomial chaos (Section 2.7.1). |
| ☐ | `.MaxInteraction` | Integer<br>default: $M$ | Maximum rank truncation: limit basis terms to MaxInteraction variables (Section 1.3.3) |
| ☐ | `.Custom` | $P \times M$ Integer array | Manual specification of the $\mathcal{A}$ index set in Eq. (1.3) |

### 3.1.2 PCE Coefficients calculation options

Method-specific options for the calculation of the PCE coefficients are reported in Tables 5 to 9.

### 3.1.3 Quadrature-specific options

| Table 5: `MetaOpts.Quadrature` | | | |
|---|---|---|---|
| ☐ | `.Level` | Integer<br>default: $p+1$ | Quadrature level |
| ☐ | `.Type` | String<br>default: dependent on $M$ | Quadrature type (full or sparse) |
| | | `'Full'` if $M < 4$ | Full tensor-product quadrature |
| | | `'Smolyak'` if $M \geq 4$ | Smolyak' sparse quadrature |
| ☐ | `.Rule` | String<br>default: `'Gaussian'` | Quadrature rule |
| | | `'Gaussian'` | Gaussian quadrature |

### 3.1.4 OLS-specific options

| Table 6: `MetaOpts.OLS` | | | |
|---|---|---|---|
| ☐ | `.TargetAccuracy` | Double<br>default: 0 | Early stop leave-one-out error threshold for degree-adaptive PCE |
| ☐ | `.KeepIterations` | Logical<br>default: false | Store additional information about OLS iterations.<br>*Warning: memory consuming.* |
| ☐ | `.ModifiedLoo` | Logical<br>default: 1 | Enable/Disable using the modified Leave-One-Out error for model selection (see Section 1.4.3) |

### 3.1.5 LARS-specific options

| Table 7: MetaOpts.LARS | | | |
|---|---|---|---|
| ☐ | .LarsEarlyStop | Logical<br>default: dependent on $N$<br><br>false if $N < 50$<br>true if $N \geq 50$ | Enable early stop during the LARS adaptive basis selection (Section 1.5.3.3). |
| ☐ | .TargetAccuracy | Double<br>default: 0 | Early stop leave-one-out error threshold. |
| ☐ | .KeepIterations | Logical<br>default: false | Store additional information about LARS iterations.<br>*Warning: memory consuming.* |
| ☐ | .HybridLoo | Logical<br>default: true | Enable/Disable applying OLS at each LARS iteration for calculating the LOO error |
| ☐ | .ModifiedLoo | Logical<br>default: 1 | Enable/Disable using the modified Leave-One-Out error for model selection (see Section 1.4.3) |

### 3.1.6 OMP-specific options

| Table 8: MetaOpts.OMP | | | |
|---|---|---|---|
| ☐ | .OmpEarlyStop | Logical<br>default: dependent on $N$<br><br>false if $N < 50$<br>true if $N \geq 50$ | Enable early stop during the OMP adaptive basis selection (Section 1.5.4.2). |
| ☐ | .TargetAccuracy | Double<br>default: 0 | Early stop leave-one-out error threshold. |
| ☐ | .KeepIterations | Logical<br>default: false | Store additional information about OMP iterations.<br>*Warning: memory consuming.* |
| ☐ | .ModifiedLoo | Logical<br>default: 1 | Enable/Disable using the modified Leave-One-Out error for model selection (see Section 1.4.3) |

### 3.1.7 SP-specific options

| Table 9: MetaOpts.SP | | | |
|---|---|---|---|
| ☐ | .NNZ | Integer<br>default: [] | User-specified value for the hyperparameter $K$ (desired number of nonzero coefficients) |

| | | | |
|---|---|---|---|
| ☐ | `.CVMethod` | String<br>default: `'loo'` | Cross-validation method for determining the hyperparameter $K$. Only used if `.NNZ` is not specified or empty. |
| | | `'loo'` | Leave-one-out cross-validation |
| | | `'kfold'` | $k$-fold cross-validation |
| ☐ | `.NumFolds` | Integer<br>default: 5 | Number of folds for $k$-fold cross-validation. Only used if `.CVMethod = 'kfold'`. |
| ☐ | `.ModifiedLoo` | Logical<br>default: 1 | Enable/Disable using the modified Leave-One-Out error for model selection (see Section 1.4.3) |

### 3.1.8 BCS-specific options

Table 10: `MetaOpts.BCS`

| | | | |
|---|---|---|---|
| ☐ | `.NumFolds` | Integer<br>default: 10 | Number of folds for $k$-fold cross-validation. |
| ☐ | `.ModifiedLoo` | Logical<br>default: 0 | Enable/Disable using the modified cross-validation error for model selection (see Section 1.4.3) |

### 3.1.9 Experimental design

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

Table 11: `MetaOpts.ExpDesign`

| | | | |
|---|---|---|---|
| ⊕ | `.Sampling` | String<br>default: `'LHS'` | 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. |

### 3.1.10 Validation Set

If a validation set is provided, UQLAB automatically calculates the validation error of the created PCE. 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 $\mathcal{Y}_{Val}$ |

### 3.1.11 Bootstrap options

Table 13: `MetaOpts.Bootstrap`

| | | | |
|---|---|---|---|
| ● | `.Replications` | Integer | Number of bootstrap replications. |

## 3.2 Accessing the results

### Syntax

```
myPCE = uq_createModel(MetaOpts) ;
```

### Output

Regardless on the configuration options given at creation time in the `MetaOpts` structure, all PCE metamodels share the same output structure, given in Table 14.

| Table 14: `myPCE` | | |
|---|---|---|
| `.Name` | String | Unique name of the PCE metamodel |
| `.Options` | Table 3 | Copy of the `MetaOpts` structure used to create the metamodel |
| `.PCE` | Table 15 | Information about all the elements of Eq. (1.3) |
| `.ExpDesign` | Table 18 | Experimental design used for calculating the coefficients |
| `.Error` | Table 19 | Error measures of the metamodelling calculation results (Section 1.4) |
| `.Internal` | Table 20 | Internal state of the MODEL object (useful for debug/diagnostics) |

### 3.2.1 Polynomial chaos expansion information

All the information needed to evaluate and post-process a PCE are contained in the `myPCE.PCE` structure. They include a basis and a set of coefficients (see Eq. (1.3)). Their format is given in Tables 15.
Note that in case the model considered has a $N_{out}$-dimensional output, each output variable $Y_i$ is treated separately and stored in `myPCE.PCE(i)`.

| Table 15: `myPCE.PCE(i)` | | |
|---|---|---|
| `.Coefficients` | $P \times 1$ Double | Truncated PCE coefficients. |
| `.Moments` | Table 16 | Post-processed moments of the PCE (Section 1.6.1). |
| `.Basis` | Table 17 | Information about the truncated polynomial basis. |

| Table 16: `myPCE.PCE(i).Moments` | | |
|---|---|---|
| `.Mean` | Double | Mean of the PCE (Eq. (1.37)) |
| `.Var` | Double | Variance of the PCE (Eq. (1.38)) |

| Table 17: `myPCE.PCE(i).Basis` | | |
|---|---|---|
| `.Degree` | Double | Maximum total polynomial degree of the basis |
| `.Indices` | $P \times M$ Double (Sparse) | Truncated set of indices $\mathcal{A}$ in Eq. (1.3) (Section 1.3) |
| `.PolyTypes` | $1 \times M$ cell array of strings | Polynomial family for each input variable. In the current version of UQLAB, each element can be one of `'Legendre'`, `'Hermite'`, `'Laguerre'` or `'Jacobi'` |
| `.MaxCompDeg` | $1 \times M$ Double | Maximum degree in each input variable of polynomials with non-zero coefficients |
| `.MaxInteractions` | Double | Maximum rank of the polynomials with non-zero coefficients |
| `.qNorm` | Double | The q-norm of the basis (1.0 unless q-norm adaptivity was used, see Section 2.7.2) |

### 3.2.2 Experimental design information

The experimental design and the corresponding model responses onto which the PCE coefficients are calculated are stored in the `myPCE.ExpDesign` structure. They are accessible as follows:

| Table 18: `myPCE.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 ($\boldsymbol{X}$ in Section 1.3.2.1) |
| `.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 |
| `.W` | $N \times 1$ Double | The Gaussian quadrature weights corresponding to each quadrature node (only available when the coefficients are calculated with the `'Quadrature'` method) |

### 3.2.3 Error estimates

The error estimates described in Sections 1.4.1 – 1.4.3 and 2.8 are available in the `myPCE.Error` output field, as described in Table 19.

| Table 19: `myPCE.Error` |
|---|

| | | |
|---|---|---|
| .LOO | Double | Leave-One-Out error (see Section 1.4.2). |
| .ModifiedLOO | Double | Modified Leave-One-Out error (see Section 1.4.3). |
| .normEmpErr | Double | Normalized Empirical Error (see Section 1.4.1) |
| .Val | Double | Validation error (see Eq. (1.17) and Section 2.8). 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 `myPCE.Internal` field. Both runtime information and complex data structures used internally by the UQLAB PCE module are stored in this structure. The general structure of the `myPCE.Internal` field is reported in Table 20. Note that not all the fields are always available, as they depend on the original configuration options.

Table 20: `myPCE.Internal`

| | | |
|---|---|---|
| .Input | INPUT object | The probabilistic input model used to build the PCE |
| .FullModel | MODEL object | Full computational model used to calculate the model response (if available) |
| .Error | Table 21 | Additional information about the PCE error estimation given in `myPCE.Error` |
| .PCE | Table 22 | Additional information on the PCE calculation. |
| .Runtime | Table 27 | Temporary variables and configuration flags used during the calculation of the PCE coefficients |

Table 21: `myPCE.Internal.Error`

| | | |
|---|---|---|
| .LOO_lars | Double | LOO error as calculated by LARS |
| .LOO_omp | Double | LOO error as calculated by OMP |

Table 22: `myPCE.Internal.PCE`

| | | |
|---|---|---|
| .Degree | Double | Final PCE degree |
| .DegreeEarlyStop | Logical | Polynomial degree early stop criterion |
| .Method | String | Algorithm used to calculate the coefficients |
| .OLS | Table 23 | OLS-specific information |
| .LARS | Table 24 | LARS-specific information |

| | | |
|---|---|---|
| `.OMP` | Table 25 | OMP-specific information |
| `.Basis` | Table 26 | Miscellaneous information about the polynomial basis (e.g., truncation parameters) |

| Table 23: myPCE.Internal.PCE.OLS | | |
|---|---|---|
| .TargetAccuracy | Double | Degree-adaptive early stop threshold |
| .LOO | Array | LOO error for each of the tested degrees in degree-adaptive mode. |
| .normEmpErr | Array | Normalized empirical error for each of the tested degrees in degree-adaptive mode. |

| Table 24: myPCE.Internal.PCE.LARS | | |
|---|---|---|
| .TargetAccuracy | Double | Degree-adaptive early stop threshold |
| .LarsEarlyStop | Logical | Early stop in LARS iterations flag |
| .HybridLoo | Logical | Enable/Disable applying OLS at each LARS iteration for calculating the LOO error |
| .ModifiedLoo | Logical | Enable/Disable the "modified LOO" error estimation in Eq. (1.20) |
| .LOO | Array | LOO error for each of the tested degrees in degree-adaptive mode. |
| .normEmpErr | Array | Normalized empirical error for each of the tested degrees in degree-adaptive mode |
| .KeepIterations | Logical | Enable storage of LARS iterations (warning: memory intensive) |
| .coeff_array | Matrix of doubles | Matrix of coefficients as calculated by each iteration of LARS (requires .KeepIterations = 1 |
| .a_scores | Vector of doubles | Array of scores for each iteration of LARS (score = 1-LOO). The final basis selected by LARS is the one with the maximum a_score |
| .loo_scores | Vector of doubles | Array of LOO erro values for each iteration of LARS |
| .lars_idx | Vector of integers | Array of indices representing the regressor chosen at each LARS iteration |

| Table 25: myPCE.Internal.PCE.OMP | | |
|---|---|---|
| .TargetAccuracy | Double | Degree-adaptive early stop threshold |
| .OmpEarlyStop | Logical | Early stop in OMP iterations flag |
| .ModifiedLoo | Logical | Enable/Disable the "modified LOO" error estimation in Eq. (1.20) |
| .LOO | Vector of doubles | LOO error for each of the tested degrees in degree-adaptive mode. |
| .normEmpErr | Vector of doubles | Normalized empirical error for each of the tested degrees in degree-adaptive mode |

| | | |
|---|---|---|
| .KeepIterations | Logical | Enable storage of OMP iterations (warning: memory intensive) |
| .coeff_array | Matrix of doubles | Matrix of coefficients as calculated by each iteration of OMP (requires .KeepIterations = 1 |
| .a_scores | Vector of doubles | Array of scores for each iteration of OMP (score = 1-LOO). The final basis selected by OMP is the one with the maximum a_score |
| .loo_scores | Vector of doubles | Array of LOO error values for each iteration of OMP |
| .omp_idx | Vector of integers | Array of indices representing the regressor chosen at each OMP iteration |

Table 26: myPCE.Internal.PCE.Basis

| | | |
|---|---|---|
| .Truncation | Structure | Structure with the truncation options used to generate the basis. See Table 4 |

Table 27: myPCE.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 | Vector of integers | 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 |
| .current_output | Integer | The current output (This is used during the calculation of the metamodel) |
| .degree_index | Integer | Index of the PCE being considered in the current status of the calculation |

# References


Babacan, S., Molina, R., and Katsaggelos, A. (2010). Bayesian compressive sensing using Laplace priors. *IEEE Trans. Image Process.*, 19(1):53–63. 16, 17, 18

Berveiller, M., Sudret, B., and Lemaire, M. (2006). Stochastic finite elements: a non intrusive approach by regression. *European Journal of Mechanics*, 15(1-3):81–92. 10

Blatman, G. (2009). *Adaptive sparse polynomial chaos expansion for uncertainty propagation and sensitivity analysis*. PhD thesis, Blaise Pascal University - Clermont II. 5, 8

Blatman, G. and Sudret, B. (2010). An adaptive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis. *Probabilistic Engineering Mechanics*, 25(2):183–197. 8

Blatman, G. and Sudret, B. (2011). Adaptive sparse polynomial chaos expansion based on Least Angle Regression. *Journal of Computational Physics*, 230:2345–2367. 11, 12

Chapelle, O., Vapnik, V., and Bengio, Y. (2002). Model selection for small sample regression. 48(1):9–23. 8

Dai, W. and Milenkovic, O. (2009). Subspace pursuit for compressive sensing signal reconstruction. *IEEE Trans. Inform. Theory*, 55(5):2230–2249. 15

Diaz, P., Doostan, A., and Hampton, J. (2018). Sparse polynomial chaos expansions via compressed sensing and D-optimal design. *Comput. Methods Appl. Mech. Engrg.*, 336:640–666. 15

Efron, B. (1979). Bootstrap methods: another look at the Jackknife. *Annals of Statistics*, 7(1):1–26. 19

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. *Annals of Statistics*, 32:407–499. 11

Ernst, O., Mugler, A., Starkloff, H.-J., and Ullmann, E. (2012). On the convergence of generalized polynomial chaos expansions. *ESAIM: Math. Model. Numer. Anal.*, 46(02):317–339. 4

Gander, W. and Gautschi, W. (2000). Adaptive quadrature revisited. *BIT Numerical Mathematics*, 40(1):84–101. 9

Gautschi, W. (1993). Is the recurrence relation for orthogonal polynomials always stable? *BIT Numerical Mathematics*, 33(2):277–284. 3

Gautschi, W. (2004). *Orthogonal polynomials: computation and approximation*. Oxford University Press on Demand. 4, 9

Gerstner, T. and Griebel, M. (1998). Numerical integration using sparse grids. *Numerical Algorithms*, 18(3-4):209–232. 10

Golub, G. H. and Welsch, J. H. (1969). Calculation of gauss quadrature rules. *Mathematics of computation*, 23(106):221–230. 9

Hastie, T., Taylor, J., Tibshirani, R., and Walther, G. (2007). Forward stagewise regression and the monotone lasso. *Electronic Journal of Statistics*, 1:1–29. 11

Ishigami, T. and Homma, T. (1990). An importance quantification technique in uncertainty analysis for computer models. In *Proc. ISUMA'90, First Int. Symp. Unc. Mod. An*, pages 398–403. University of Maryland. 21

Lüthen, N., Marelli, S., and Sudret, B. (2020). Sparse polynomial chaos expansions: Literature survey and benchmark. *SIAM/ASA J. Unc. Quant.* (submitted). 15, 17

Mallat, S. and Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415. 13

Marelli, S. and Sudret, B. (2018). An active-learning algorithm that combines sparse polynomial chaos expansions and bootstrap for structural reliability analysis. *Struct. Saf.*, 75:67–74. 19

Migliorati, G., Nobile, F., Von Schwerin, E., and Tempone, R. (2013). Approximation of quantities of interest in stochastic PDEs by the random discrete L2 projection on polynomial spaces. *SIAM Journal of Scientific Computing*, 35(3):1440–1460. 11

Pati, Y., Rezaiifar, R., and Krishnaprasad, P. (1993). Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA*, pages 40–44. 13

Sargsyan, K., Safta, C., Najm, H., Debusschere, B., Ricciuto, D., and Thornton, P. (2014). Dimensionality reduction for complex models via Bayesian compressive sensing. *Int. J. Uncertain. Quantificat.*, 4(1):63–93. 16

Shampine, L. (2008). Vectorized adaptive quadrature in MATLAB. *Journal of Computational and Applied Mathematics*, 211(2):131–140. 4

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. 2

Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288. 11

Tipping, M. E. (2001). Sparse Bayesian learning and the relevance vector machine. *J. Mach. Learn. Res.*, 1(Jun):211–244. 16

Tipping, M. E. and Faul, A. C. (2003). Fast marginal likelihood maximisation for sparse Bayesian models. In *Proc. 9th Int. Workshop on Artificial Intelligence and Statistics*. 18

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. 3