ETH zürich

# UQLab User Manual
# Polynomial Chaos Expansions

S. Marelli, B. Sudret

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

Risk, Safety &
Uncertainty Quantification

**How to cite UQLAB**

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

**How to cite this manual**

S. Marelli and B. Sudret, UQLab user manual – Polynomial Chaos Expansions, Report UQLab-V1.0-104, Chair of Risk, Safety & Uncertainty Quantification, ETH Zurich, 2017.

**BIBTEX entry**

```
@TECHREPORT{UQdoc_10_104,,
author = {Marelli, S. and Sudret, B.},
title = {UQLab user manual -- Polynomial Chaos Expansions},
institution = {Chair of Risk, Safety \& Uncertainty Quantification, ETH Zurich},
year = {2017},
note = {Report UQLab-V1.0-104}
}
```

**List of contributors:**

*M. Berchier*: Implementation and documentation of the orthogonal matching pursuit (OMP) regression method

# Document Data Sheet

| | |
|---|---|
| Document Ref. | UQLab-V1.0-104 |
| Title: | UQLab User Manual– Polynomial Chaos Expansions |
| Authors: | S. Marelli, B. Sudret<br>Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland |
| Date: | 01/05/2017 |

| Doc. Version | Date | Comments |
|---|---|---|
| V. 1.0 | 01/05/2017 | UQLab v1.0 release<br>• added orthogonal matching pursuit<br>• added arbitrary polynomials support<br>• minor fixes to formatting |
| V. 0.9 | 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 1.3.2.

## 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:

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

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 multiplicands.

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

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

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 Families of univariate orthonormal polynomials

#### 1.3.1.1 Askey-Scheme 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 description

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

of each of the classical families of 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 1.7.

**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.

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

It is possible to construct sets of univariate polynomials of the form: $\tilde{\pi}_n = \frac{\pi_n}{\sqrt{\langle \pi_n, \pi_n \rangle}}$ orthogonal with respect to an arbitrary probability distribution $f(x)$. In UQLAB this is performed by employing the Stieltjes procedure. The Stieltjes procedure employs a recurrence relation that holds for the computation of any orthogonal polynomial that reads:

$$\sqrt{\beta_{n+1}}\tilde{\pi}_{n+1}(x) = (x-\alpha_n)\tilde{\pi}_n(x) - \sqrt{\beta_n}\tilde{\pi}_{n-1}(x), \quad n = 0, 1, 2, \dots \tag{1.7}$$

and the so-called *Christoffel-Darboux* formulae

$$\alpha_n = \frac{\langle x\pi_n, \pi_n \rangle}{\langle \pi_n, \pi_n \rangle} \tag{1.8}$$

$$\beta_n = \frac{\langle \pi_n, \pi_n \rangle}{\langle \pi_{n-1}, \pi_{n-1} \rangle} \tag{1.9}$$

where $\langle g, h \rangle = \int g(x)h(x)f(x)dx$. 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).

### 1.3.2 Basis truncation schemes

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

$$\mathcal{A}^{M,p} = \{\boldsymbol{\alpha} \in \mathbb{N}^M \ : \ |\boldsymbol{\alpha}| \leq p\} \qquad \text{card } \mathcal{A}^{M,p} \equiv P = \binom{M+p}{p} \tag{1.10}$$

Several additional truncation schemes can be built that are better suited to various types of applications. Within UQLAB, two additional non-mutually-exclusive schemes are available: maximum interaction and hyperbolic norm.

#### 1.3.2.1 Maximum interaction

This truncation scheme is based on choosing a subset of the terms defined in Eq. (1.10), such that the $\boldsymbol{\alpha}$'s have at most $r$ non-zero elements (low-rank $\boldsymbol{\alpha}$):

$$\mathcal{A}^{M,p,r} = \{\boldsymbol{\alpha} \in \mathcal{A}^{M,p} \ : \ ||\boldsymbol{\alpha}||_0 \leq r\}, \tag{1.11}$$

where $||\boldsymbol{\alpha}||_0 = \sum_{i=1}^{M} \mathbf{1}_{\{\alpha_i > 0\}}$ is the *rank* of the multi-index $\boldsymbol{\alpha}$.
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.

#### 1.3.2.2 Hyperbolic truncation

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

$$\mathcal{A}^{M,p,q} = \{\boldsymbol{\alpha} \in \mathcal{A}^{M,p} \ : \ ||\boldsymbol{\alpha}||_q \leq p\}, \tag{1.12}$$

where:

$$||\boldsymbol{\alpha}|| = \left(\sum_{i=1}^{M} \alpha_i^q\right)^{1/q}. \tag{1.13}$$

Note that for $q = 1$ hyperbolic truncation corresponds exactly to the standard truncation scheme in Eq. (1.10). For $q < 1$, hyperbolic truncation includes all the high-degree terms in each single variable, but discourages equivalently high order interaction terms. An example of the behaviour of the hyperbolic norm in two dimensions for different values of $p$ and $q$ is shown in Figure 1.

### 1.3.3 Extension to arbitrary joint input distributions

In case the input random variables are not independent, or no standard polynomials are defined for their marginal distributions, it is possible to define an isoprobabilistic transform from the original probabilistic space to the so-called *reduced space*. 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:

$$\boldsymbol{X} = \mathcal{T}(\boldsymbol{Z}), \qquad \boldsymbol{Z} = \mathcal{T}^{-1}(\boldsymbol{X}) \tag{1.14}$$

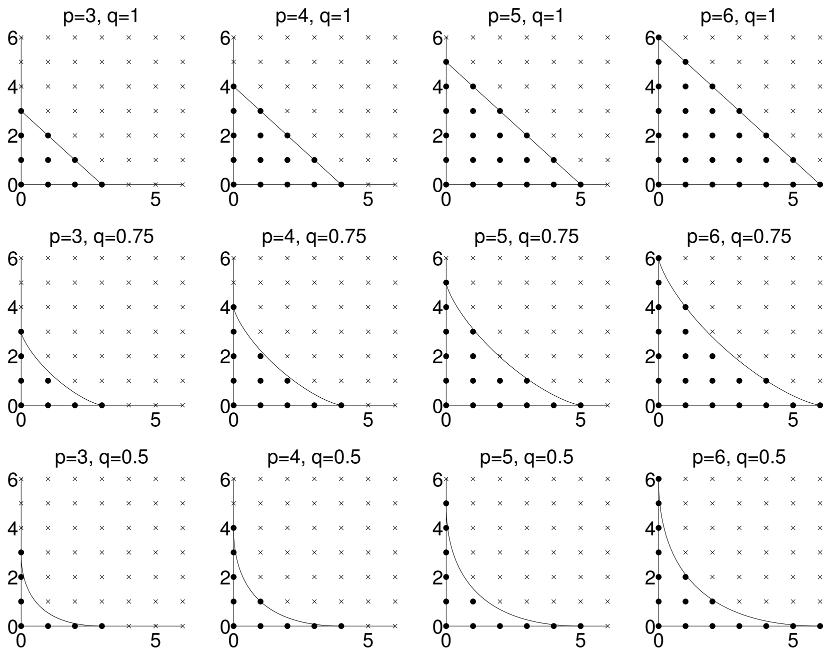


Figure 1: Hyperbolic truncation set for varying values of $p$ (constant in each column) and $q$ (constant in each row) as defined in Eq. (1.12). For $q = 1$, hyperbolic truncation reduces to the standard truncation scheme (first row). Decreasing the value of $q$ decreases the amount of mixed order polynomials included in the expansion.

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

$$Y = \mathcal{M}(\boldsymbol{Z}) = \sum_{\boldsymbol{\alpha} \in \mathbb{N}^M} y_{\boldsymbol{\alpha}} \Psi_{\boldsymbol{\alpha}} \left( \mathcal{T}\left( \boldsymbol{Z} \right) \right) \tag{1.15}$$

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.

## 1.4 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 the result of the post-processing of a set of model evaluations, the *experimental design*, that are given on a proper sampling of the input random variables.

The two principal strategies to calculate the polynomial chaos coefficients non-intrusively are projection and least-square minimization.

### 1.4.1 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:

$$y_{\boldsymbol{\alpha}} = \mathbb{E}\left[\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X}) \cdot \mathcal{M}(\boldsymbol{X})\right] \tag{1.16}$$

The calculation of the coefficients is therefore reduced to the calculation of the expectation value in Eq. (1.16). 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:

$$y_{\boldsymbol{\alpha}} = \int_{\Omega_{\boldsymbol{X}}} \mathcal{M}(\boldsymbol{x})\Psi_{\boldsymbol{\alpha}}(\boldsymbol{x}) f_{\boldsymbol{X}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} \approx \sum_{i=1}^{N} w^{(i)} \mathcal{M}(\boldsymbol{x}^{(i)})\Psi_{\boldsymbol{\alpha}}(\boldsymbol{x}^{(i)}) \tag{1.17}$$

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 (1.17). 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. (1.17), but the weights are derived from a combination of lower order standard quadrature terms:

$$Q^{M,l}_{Smolyak} \equiv \sum_{l+1 \leqslant |\boldsymbol{i}| \leqslant l+M} (-1)^{M+l-|\boldsymbol{i}|} \cdot \begin{pmatrix} M-1 \\ k+M-|\boldsymbol{i}| \end{pmatrix} \cdot Q^{\boldsymbol{i}} \tag{1.18}$$

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.

#### 1.4.1.1 Error estimation

An *a posteriori* error estimate of the Gaussian quadrature error in the estimation of the PCE coefficients in Eq. (1.17) 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:

$$\epsilon_{res} \approx \frac{\sum_{i=1}^{N} \left[w^{(i)} \left(Y^{(i)} - \boldsymbol{y}^{\mathsf{T}} \boldsymbol{\Psi}(\boldsymbol{x}^{(i)})\right)\right]^2}{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \hat{\mu}_Y\right)^2} \tag{1.19}$$

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.

### 1.4.2 Least-Squares minimization method

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:

$$Y = \mathcal{M}(\boldsymbol{X}) = \sum_{j=0}^{P-1} y_j \, \Psi_j(\mathbf{X}) + \varepsilon_P \equiv \boldsymbol{y}^{\mathsf{T}} \Psi(\boldsymbol{X}) + \varepsilon_P \tag{1.20}$$

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 matrix that assembles the values of all the orthonormal polynomials in $\boldsymbol{X}$.

The least-square minimization problem can then be setup as:

$$\hat{\boldsymbol{y}} = \arg\min \mathbb{E}\left[\left(\boldsymbol{y}^{\mathsf{T}} \Psi(\boldsymbol{X}) - \mathcal{M}(\boldsymbol{X})\right)^2\right]. \tag{1.21}$$

#### 1.4.2.1 Ordinary Least-Squares

A direct approach to solving Eq. (1.21) is given by Ordinary Least-Squares (OLS). Given a sampling of size $N$ of the input random vector $\boldsymbol{\mathcal{X}} = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}\}^{\mathsf{T}}$ (the experimental design) and the corresponding model responses $\mathcal{Y} = \{y^{(1)}, ..., y^{(N)}\}^{\mathsf{T}}$, the ordinary least-square solution of Eq. (1.21) reads:

$$\hat{\boldsymbol{y}} = (\mathbf{A}^{\mathsf{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathsf{T}}\mathcal{Y}, \tag{1.22}$$

where:

$$A_{ij} = \Psi_j\left(\boldsymbol{x}^{(i)}\right) \quad i = 1, \dots, n \; ; \; j = 0, \dots, P-1 \tag{1.23}$$

is the so-called experimental matrix that contains the values of all the basis polynomials in the experimental design points.

The main advantage of the least-square minimization 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 $\boldsymbol{X}$. Recent theoretical background to the convergence of the least-square minimization method can be found in Migliorati et al. (2013).

### 1.4.3 *A posteriori* error estimation

Due to its formulation in Eq. (1.20), least-square minimization offers a natural candidate to estimate the residual error $\epsilon_P$. The relative generalization error $\epsilon_{gen}$ is defined as:

$$\epsilon_{gen} = \mathbb{E}\left[\left(\mathcal{M}(\boldsymbol{X}) - \mathcal{M}^{PC}(\boldsymbol{X})\right)^2\right] / \mathrm{Var}\left[Y\right] \tag{1.24}$$

There are two main ways to estimate $\epsilon_{gen}$: via *normalized empirical error* and *leave-one-out cross-validation error.*

#### 1.4.3.1 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:

$$\epsilon_{emp} = \frac{\sum_{i=1}^{N}\left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \mathcal{M}^{PC}(\boldsymbol{x}^{(i)})\right)^2}{\sum_{i=1}^{N}\left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \hat{\mu}_Y\right)^2} \tag{1.25}$$

where $\hat{\mu_Y} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{M}(\boldsymbol{x}^{(i)})$ is the sample mean of the experimental design response.

The estimator in Eq. (1.25), 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.

#### 1.4.3.2 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:

$$\epsilon_{LOO} = \frac{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \mathcal{M}^{PC\setminus i}(\boldsymbol{x}^{(i)})\right)^2}{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \hat{\mu}_Y\right)^2}. \tag{1.26}$$

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

$$\epsilon_{LOO} = \sum_{i=1}^{N} \left(\frac{\mathcal{M}(\boldsymbol{x}^{(i)}) - \mathcal{M}^{PC}(\boldsymbol{x}^{(i)})}{1 - h_i}\right)^2 \Big/ \sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \hat{\mu}_Y\right)^2, \tag{1.27}$$

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

$$\boldsymbol{h} = \text{diag}\left(\mathbf{A}(\mathbf{A}^{\mathsf{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathsf{T}}\right), \tag{1.28}$$

and $\mathbf{A}$ is the experimental matrix in Eq. (1.23).

#### 1.4.3.3 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:

$$\epsilon^* = \epsilon\, T(P, N), \tag{1.29}$$

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:

$$T(P, N) = \frac{N}{N - P}\left(1 + \frac{\text{tr}(\mathbf{C}_{emp}^{-1})}{N}\right), \tag{1.30}$$

with

$$\mathbf{C}_{emp} = \frac{1}{N}\mathbf{A}^{\mathsf{T}}\mathbf{A}. \tag{1.31}$$

## 1.5 Adaptive sparse PCE

*A-posteriori* error estimation is a very important tool at the basis of any adaptive algorithm, because it allows one to estimate the accuracy of the model without the need of running additional expensive model evaluations to generate a proper validation set. In the context of PCE, it can be used in two main families of adaptive algorithms :

- **Basis-adaptive PCE:** starting from a small candidate polynomial basis, one gradually adds new elements (*e.g.*by increasing the maximum polynomial degree in the truncation scheme) and calculates the corresponding PCE. The best one in terms of generalization error is chosen as the best candidate.
- **Sparse PCE:** starting from a candidate polynomial basis, only a subset of the most relevant polynomials is retained, while the coefficients of all the others are set to 0.

The two families of adaptivity are not mutually exclusive: Blatman and Sudret (2011) propose an algorithm that combines both basis-adaptivity by iteratively increasing the polynomial degree and sparse basis selection at each iteration to minimize over-fitting.

### 1.5.1 Sparse PCE: Least Angle regression

One commonly used strategy for basis-adaptive PCE is that of iteratively relaxing the truncation scheme by increasing the maximum allowed polynomial degree. Given a target accuracy $\epsilon_T$ and a maximum number of iterations $NI_{max}$, the algorithm can be summarized as follows:

1. Generate an initial basis with one or a combination of the truncation schemes in Section 1.3.2, with $p = p_0$;
2. Calculate the PCE coefficients and the corresponding generalization error estimate $\epsilon_{LOO}$ in Eqs. (1.26) or (1.29) by least-square minimization;
3. Compare the error with a preset threshold $\epsilon_T$. If $\epsilon_{LOO} \leq \epsilon_T$ or the number of iterations $NI = NI_{max}$, stop the algorithm and return the PCE with the lowest generalization error. Otherwise, set $p = p + 1$ and return to Step 1.

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

### 1.5.2 Sparse PCE: Least Angle Regression

In most applied science problems, only low order interactions between the input variables tend to be important. This is known as the *sparsity-of-effects principle*. In other words, when choosing adaptive PCE calculation strategies, models that favour low rank truncation schemes should be preferred. The truncation schemes presented in Section 1.3.2.1 and 1.3.2.2 both produce sparser PCE than the classical truncation scheme.

A complementary strategy to favour sparsity in high dimension consists in directly modifying the least-square minimization problem in Eq. (1.21) by adding a penalty term of the form $\lambda||\boldsymbol{y}||_1$, *i.e.* solving:

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathbb{R}^{|\mathcal{A}|}}{\operatorname{argmin}} \, \mathbb{E}\left[\left(\boldsymbol{y}^{\mathsf{T}} \Psi(\boldsymbol{X}) - \mathcal{M}(\boldsymbol{X})\right)^2\right] + \lambda||\boldsymbol{y}||_1, \tag{1.32}$$

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. (1.32), 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.

#### 1.5.2.1 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.
Because the algorithm produces an ordinary least-squares solution to a reduced-size regression problem,
the *a posteriori* error estimate in Eq. (1.27) can be used as a measure of the accuracy of the current model, hence enabling adaptivity.
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 $\Psi_{\alpha_j}$ 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.

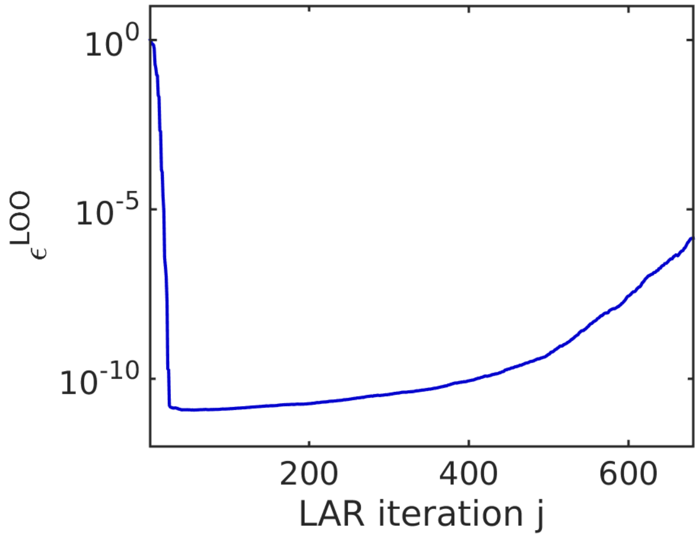


Figure 2: Typical evolution of $\epsilon_{LOO}$ vs. LAR iteration. The evolution is in many cases smooth and consistently convex throughout the iterations. In some cases with very small experimental designs, however, local minima in the early iterations can be observed.

#### 1.5.2.2 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). This limitation can be overcome by introducing the so-called *hybrid-LAR* step. At the end of the LAR iterations, after the best basis is selected, the constant regressor is added to the selected basis and OLS is performed to calculate the final coefficients.

#### 1.5.2.3 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 $\epsilon_{LOO}^j$ 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.

### 1.5.3 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:

$$R_n = \left\langle R_n, \Psi_{\boldsymbol{\alpha}_{n+1}} \right\rangle \Psi_{\boldsymbol{\alpha}_{n+1}} + R_{n+1} \tag{1.33}$$

Therefore, the residual $R_{n+1}$ is by construction orthogonal to the new polynomial $\Psi_{\boldsymbol{\alpha}_{n+1}}$. Projecting Eq. (1.33) onto $R_n$ yields:

$$\left\| R_n \right\|^2 = \left| \langle R_n, \Psi_{\boldsymbol{\alpha}_{n+1}} \rangle \right|^2 + \left\| R_{n+1} \right\|^2 . \tag{1.34}$$

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:

$$\Psi_{\boldsymbol{\alpha}_{n+1}} = \arg \max_{\boldsymbol{\alpha} \in \mathcal{A}} \left| \langle R_n, \Psi_{\boldsymbol{\alpha}} \rangle \right| \tag{1.35}$$

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*.

#### 1.5.3.1 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. (1.26) 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.

#### 1.5.3.2 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 $\epsilon^j_{LOO}$ 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.

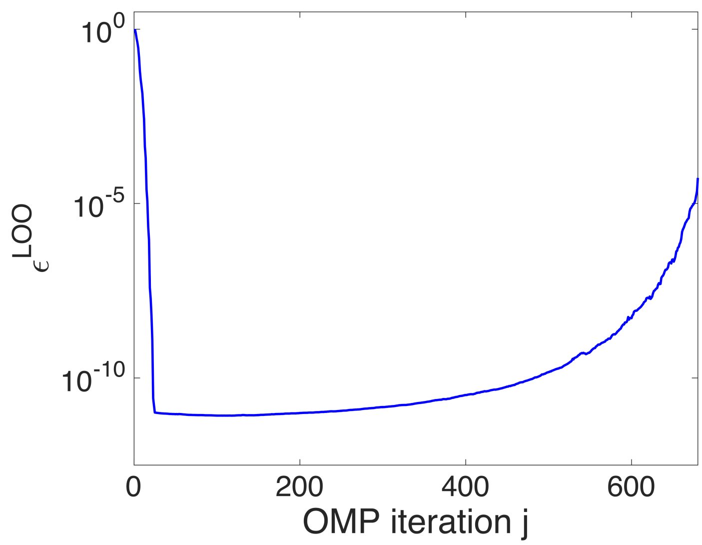


Figure 3: Typical evolution of $\epsilon_{LOO}$ vs. OMP iterations. The evolution is in many cases relatively smooth and consistently convex throughout the iterations. In some cases with very small experimental designs, however, local minima in the early iterations can be observed.

## 1.6 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:

$$\mu^{PC} = \mathbb{E}\left[\mathcal{M}^{PC}(\boldsymbol{X})\right] = y_{\mathbf{0}} \tag{1.36}$$

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

$$(\sigma^{PC})^2 = \mathbb{E}\left[(\mathcal{M}^{PC}(\boldsymbol{X}) - \mu^{PC})^2\right] = \sum_{\substack{\boldsymbol{\alpha}\in\mathcal{A} \\ \boldsymbol{\alpha}\neq\mathbf{0}}} y_{\boldsymbol{\alpha}}^2 \tag{1.37}$$

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

## 1.7 Other post-processing techniques

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.

# 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 setup

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 setup 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, please 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, please refer to the UQLAB User Manual – the INPUT module .

## 2.3 Setup 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 subsections.

## 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. For non-classical input distributions, the univariate orthonormal polynomials are computed numerically by applying the Stieltjes procedure. See Section 1.3.1.2.

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 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 truncation scheme in Eq. (1.10), 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 1.3.2.1 and 1.3.2.2 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.Degree = 2;
MetaOpts.TruncOptions.Custom = [0 0 0; 0 2 0; 1 0 0; 0 1 1];
```

## 2.5 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 1.4 are deployed in UQLAB.

### 2.5.1 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:    26384
Quadrature error:          5.7065916e-10
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 4.

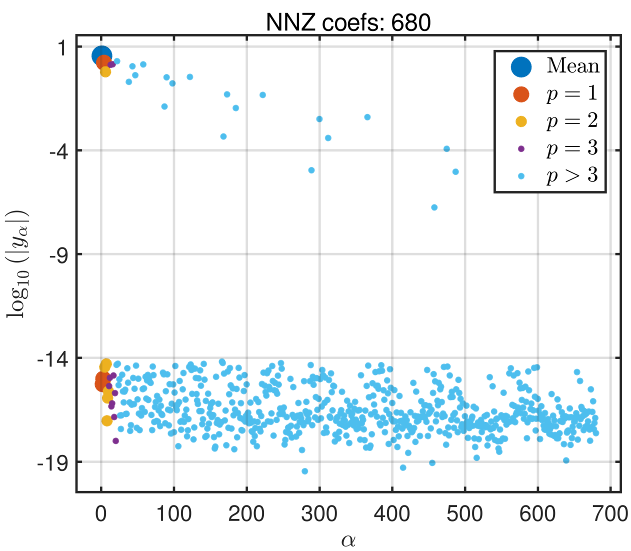


Figure 4: Graphical representation of the logarithmic spectrum of the PCE coefficients. Most of the coefficients of the 680 basis elements are close to 0.

#### 2.5.1.1 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 $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X})$ basis elements. The corresponding basis elements are given by the `PCE.Basis` structure:

```
myPCE_Quadrature.PCE.Basis
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. (1.12)). 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. (1.36),(1.37)).

For more details about the available information in the PCE output, please 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. (1.17).

***A posteriori* error estimates**

The structure `myPCE_Quadrature.Error` contains the normalized quadrature error estimate from equation Eq. (1.19).

#### 2.5.1.2 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 1.4.1 and specifying the quadrature level.

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

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

  Please 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.

### 2.5.2 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:          1.9344452e-08
Mean value:                    3.5000
Standard deviation:            3.7209
Coef. of variation:          106.310%
%--------------------------------------------------%
```

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 5.

#### 2.5.2.1 Accessing the results

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

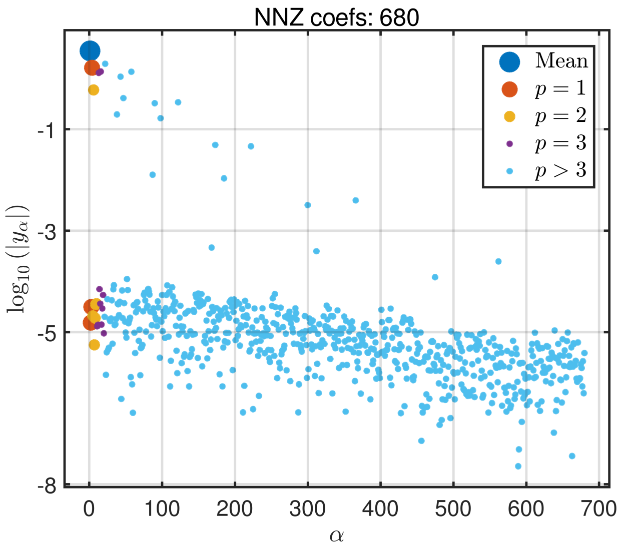


Figure 5: Graphical representation of the logarithmic spectrum of the PCE coefficients. Again, most of the coefficients of the 680 basis elements are close to 0.

```
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 2.5.2.2 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: 1.9344e-08
  normEmpError: 1.2210e-12
```

where `Error.normEmpError` and `Error.LOO` corresponds to the empirical error estimate in Eq. (1.25) and to the modified leave-one-out error in Eq. (1.27), respectively.
Note that the `Error.normEmpError` 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 19.

#### 2.5.2.2 Advanced options

There are no OLS specific options *per se*. However, when combined with *degree adaptive PCE* described in Section 1.5 and 2.5.6, 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 1.5. 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. (1.30) and the correction factor in (1.31). It is possible, however, to enable the classical LOO estimator in Eq. (1.26) 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 2.5.5. A detailed list of the available configuration options for OLS can be found in Table 6.

### 2.5.3 Sparse PCE: Least Angle Regression

Configuring a sparse PCE with least angle regression (LARS) is very similar to setting up a PCE with OLS (they are both regression-based methods). The code needed to create a basic LARS-based PCE with 1,000 samples 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_LARS = uq_createModel(MetaOpts);
```

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

```
uq_print(myPCE_LARS);
```

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:         47
   Full model evaluations:     1000
   Leave-one-out error:          9.7793015e-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_LARS);
```

which produces the image in Figure 6. Note how the LARS solution only produces 47 non-

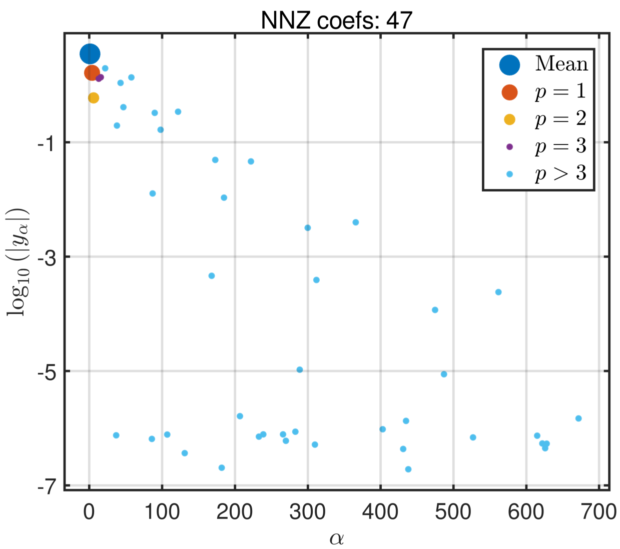


Figure 6: Graphical representation of the logarithmic spectrum of the PCE coefficients. The sparsity of the LARS solution w.r.t. its full counterpart OLS in Figure 5 is clear.

zero coefficients, w.r.t. to the 680 of its non-sparse counterpart OLS, resulting in a much sparser spectrum in Figure 6.

#### 2.5.3.1 Accessing the results

**Coefficients and basis** The coefficients and basis can be accessed from the structure `myPCE_LARS.PCE`:

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

**Model evaluations**

The experimental design structure containing the model evaluations used to calculate the PCE coefficients is identical to that in OLS in Section 2.5.1.1.

```
myPCE_LARS.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 2.5.5 for advanced options for the creation of the experimental design.

***A posteriori* error estimates**

The *a posteriori* error estimates are stored in the `myPCE_LARS.Error` structure in the same format as in Section 2.5.2.1:

```
myPCE_LARS.Error
ans =
  LOO: 9.7793e-12
  normEmpError: 8.5794e-12
```

Note that the LOO error for sparse PCE is significantly smaller than for OLS ($\approx 10^{-8}$), even though the experimental design has the same size $N = 1,000$.
For a comprehensive overview of the outputs available for the LARS method, see Table 20.

#### 2.5.3.2 Advanced options

Albeit the default settings are optimal for most real case scenarios, the LARS method allows for the customization of several parameters that can be used to fine-tune the coefficients estimation. They 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 1.5.2.3, add:

```
MetaOpts.LARS.LarsEarlyStop = 0;
```

Note that this option can significantly increase the computational time necessary to calculate the coefficients.

- **Disable hybrid LARS:** the final OLS step of Hybrid LARS can be disabled as follows:

```
MetaOpts.LARS.HybridLars = 0;
```

If Hybrid LARS is disabled, the first coefficient is set to the mean of the experimental design: $y_0 = \hat{\mu}_Y$.

- **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_LARS.Internal.PCE.LARS.coeff_array`.

- **Disable the modified LOO estimator:** by default, $\epsilon_{LOO}$ is calculated with the modified estimator in Eq. (1.30) and the correction factor in (1.31). It is possible, however, to enable the classical LOO estimator in Eq. (1.26) as follows:

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

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

A detailed list of the available configuration options for LARS can be found in Table 7. 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 2.5.5.

### 2.5.4 Sparse PCE: Orthogonal Matching Pursuit

Configuring a sparse PCE using orthogonal matching pursuit (OMP) is very similar to setting up a PCE with other regression methods (OLS or LARS). The code needed to create a OMP-based PCE with 1000 samples in the experimental design is as follows:

```
% Reminder of previous configuration options
MetaOpts.Type = 'Metamodel';
MetaoOpts.MetaType = 'PCE';

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

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

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

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

```
uq_print(myPCE_OMP);
```

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:             100

Full model evaluations:           1000
Leave-one-out error:              8.3177894e-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 be visualized graphically as follows:

```
uq_display(myPCE_OMP);
```

which produces the image in Figure 7. The solution with OMP produces only 118 non-zero coefficients, w.r.t. the 680 of its non-sparse counterpart OLS, resulting in the much sparser spectrum shown in Figure 7. The OMP solution tends to produce less sparse solutions for this particular model with respect to LARS (cf. Figure 6).

#### 2.5.4.1 Accessing the results

**Coefficients and basis** The coefficients and basis can be accessed from the structure myPCE_OMP.PCE:

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

**Model evaluations**

The experimental design structure containing the model evaluations used to calculate the PCE coefficients is identical to that in OLS in Section 2.5.1.1.

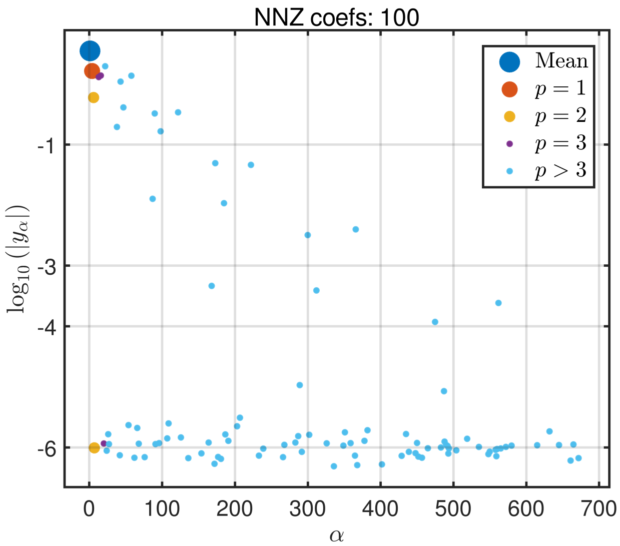


Figure 7: Graphical representation of the logarithmic spectrum of the PCE coefficients. The sparsity of the OMP solution w.r.t. its full counterpart OLS in Figure 5 is clear.

```
myPCE_OMP.ExpDesign
ans =
NSamples: 1000
Sampling: 'LHS'
ED_Input: [1x1 uq_input]
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 2.5.5 for advanced options for the creation of the experimental design.

### *A posteriori* error estimates

The *a posteriori* error estimates are stored in the `myPCE_OMP.Error` structure in the same format as in Section 2.5.2.1:

```
myPCE_OMP.Error
ans =
LOO: 8.3178e-12
normEmpError: 6.4145e-12
```

Note that the LOO error for sparse PCE is significantly smaller than for OLS ($\approx 10^{-8}$), even though the experimental design has the same size $N = 1,000$.

For a comprehensive overview of the outputs available for the OMP method, see Table 21.

#### 2.5.4.2 Advanced options

Albeit the default settings are optimal for most real case scenarios, the OMP method allows for the customization of several parameters that can be used to fine-tune the coefficients estimation. They 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 1.5.3.2, add:

```
MetaOpts.OMP.OmpEarlyStop = 0;
```

  Note that this option can significantly increase the computational time necessary to calculate the coefficients.

- **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`.

- **Disable the modified LOO estimator:** by default, $\epsilon_{LOO}$ is calculated with the modified estimator in Eq. (1.30) and the correction factor in Eq. (1.31). It is possible, however, to enable the classical LOO estimator in Eq. (1.26) as follows:

```
MetaOpts.OMP.ModifiedLoo = 0;
```

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

A detailed list of the available configuration options for OMP can be found in Table 8. 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 2.5.5.

### 2.5.5 Advanced experimental design options

Several 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:** by default, the experimental design is sampled with latin hypercube sampling (LHS). It is possible to specify any specific 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 in the configuration with the same format as in Section 2.5.1.1. Assuming such values are stored in the local variables `X_ED` and `Y_ED`, they can be imported in UQLAB as follows:

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

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

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

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

  **Important note:** When an experimental design is specified manually there is no need to create a MODEL object as in Section 2.2. 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 9.

### 2.5.6 Degree-Adaptive PCE

Least-square minimization methods (OLS and LARS) provide the $\epsilon_{LOO}$ error estimator, which can be enabled to develop basis-adaptive PCE as described in Section 1.5. 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: 7.5708e-19
  normEmpError: 3.7798e-20
```

The resulting coefficients spectrum is shown in Figure 8.

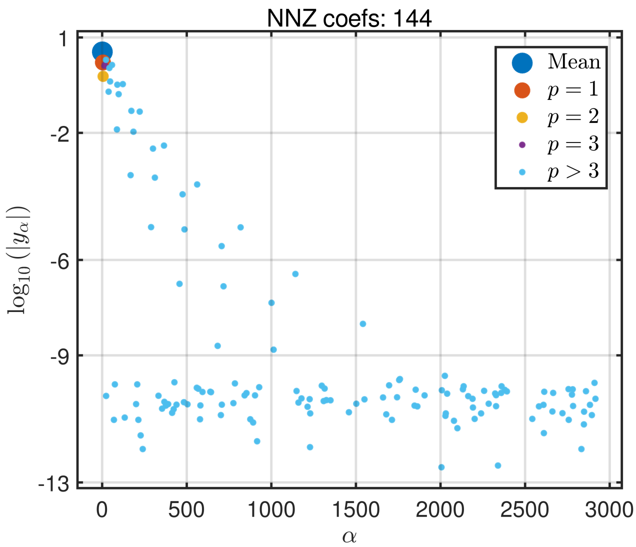


Figure 8: Graphical representation of the logarithmic spectrum of the PCE coefficients for degree-adaptive PCE. The analysis converged to $p = 24$ with $N = 256$.

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.

#### 2.5.6.1 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.

#### 2.5.6.2 Advanced options

The default behaviour of the degree-adaptive scheme is to automatically stop increasing the maximal degree if the $\epsilon_{LOO}$ has not decreased for at least two 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 = 0;
```

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 increase dramatically 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.

## 2.6 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;
```

## 2.7 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`

### 2.7.1 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 2.5.1.1, 2.5.2.1 and 2.5.3.1.

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

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

## 2.8 Using PCE as a model (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 ).

Consider the Ishigami example in Section 2.1. After calculating the coefficients with any of the methods described in Section 2.5, 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, please 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 9. Due to the high accuracy of the model, the original function and the metamodel are virtually indistinguishable.

## 2.9 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 2.8. This allows, e.g., to import a metamodel

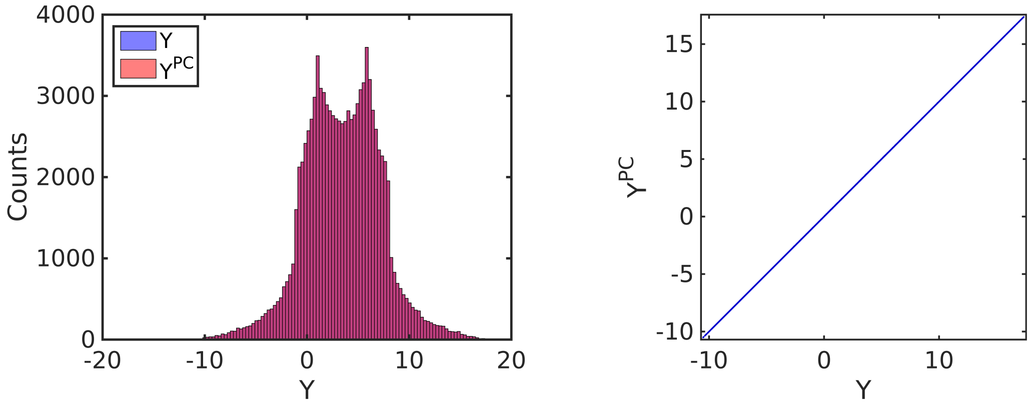


Figure 9: Histogram and scatter plots of true *vs.* metamodelled responses of the Ishigami function to a sample of the input of size $n = 10^5$.

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.

## 2.10 Using PCE with constant parameters

In some analyses, one may need to assign a constant value to one or to a set of parameters. When this is the case, the PCE metamodel is built by internally removing the constant parameters from the inputs. This process is transparent to the users as they shall still evaluate the model using the full set of parameters (including those which were set constant). UQLAB will automatically and appropriately account for the set of input parameters which were declared constant.
To set a parameter to constant, the following command can be used (See UQLAB User Manual – the INPUT module ):

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

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

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

# Chapter 3

# Reference List

### How to read the reference list

Structures play an important role throughout the UQLAB syntax. They offer a natural way to group configuration options and output quantities semantically. Due to the complexity of the algorithms implemented, it is not uncommon to employ nested structures to fine-tune inputs/outputs. Throughout this reference guide, we adopt a table-based description of the configuration structures.
The simplest case is given when a field of the structure is a simple value/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 correspond to name, 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, we provide a link to a table that describes the available options, 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/doubles mean a real number represented in double precision (resp. a set of such real numbers).

## 3.1 Create a PCE meta-model

### Syntax

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

### Input

The struct variable `MetaOpts` contains the following fields:

Table 3: `MetaOpts`

| | | | |
|---|---|---|---|
| ● | `.Type` | `'uq_metamodel'` | Select the metamodelling tool |
| ● | `.MetaType` | `'PCE'` | Select polynomial chaos expansion |
| □ | `.Input` | INPUT object | Probabilistic input model (See Section 2.6) |
| □ | `.Name` | String | Unique identifier for the meta-model |
| □ | `.Display` | String | 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 2.5.6 |
| □ | `.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 1.3.2) |
| □ | `.Method` | String<br>default: `'LARS'` | Coefficients calculation method (Section 2.5) |

| | | | |
|---|---|---|---|
| | | 'Quadrature' | Quadrature (Section 1.4.1) |
| | | 'OLS' | Ordinary Least-Squares Regression (Section 1.4.2.1) |
| | | 'LARS' | Least-Angle Regression (Section 1.5.2) |
| | | 'OMP' | Orthogonal Matching Pursuit (Section 1.5.3) |
| | | 'Custom' | User-defined PCE coefficients and basis (no calculations) |
| ⊞ | .Quadrature | Table 5 | Quadrature options (Section 2.5.1.2) |
| ⊞ | .OLS | Table 6 | OLS-specific options (Section 2.5.2.2) |
| ⊞ | .LARS | Table 7 | LARS-specific options (Section 2.5.3) |
| ⊞ | .OMP | Table 8 | OMP-specific options (Section 2.5.4) |
| ⊞ | .PCE | Table 11 | Custom-PCE parameters (Section 2.9). Use the same format as the default output of the calculation |
| ☐ | .FullModel | MODEL object | UQLab model used to create an experimental design (Section 2.6) |
| ☐ | .ExpDesign | Table 9 | Experimental design-specific options (Section 2.5.5) |

### 3.1.1 Truncation options

The truncation strategies described in Section 1.3.2 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 1.3.2). Corresponds to $0 < q \leq 1$ in Eq. (1.12) |
| ☐ | .MaxInteraction | Integer<br>default: $M$ | Maximum rank truncation: limit basis terms to MaxInteraction variables (Section 1.3.2) |
| ☐ | .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 7.

### 3.1.3 Quadrature-specific options

| Table 5: MetaOpts.Quadrature | | | |
|---|---|---|---|
| ☐ | .Level | Integer<br>default: $p + 1$ | Quadrature level |
| ☐ | .Type | String<br>default: 'Smolyak' | Quadrature type (full or sparse) |
| | | 'Full' | Full tensor-product quadrature |
| | | 'Smolyak' | 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 |

### 3.1.5 LARS-specific options

| Table 7: MetaOpts.LARS | | | |
|---|---|---|---|
| ☐ | .LarsEarlyStop | Logical<br>default: true | Enable early stop during the LARS adaptive basis selection (Section 1.5.2.3). |
| ☐ | .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.* |
| ☐ | .HybridLars | Logical<br>default: true | Enable/Disable hybrid LARS (Section 1.5.2.2) |

### 3.1.6 OMP-specific options

| Table 8: MetaOpts.OMP | | | |
|---|---|---|---|
| ☐ | .OmpEarlyStop | Logical<br>default: true | Enable early stop during the OMP adaptive basis selection (Section 1.5.3.2). |
| ☐ | .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.* |

### 3.1.7 Experimental design

If a model is specified, UQLAB can automatically create an experimental design for PCE. The available options are listed in Table 9.

Table 9: `MetaOpts.ExpDesign`

| | | | |
|---|---|---|---|
| ⊕ | `.Sampling` | String<br>default: `'MC'` | Sampling type |
| | | `'MC'` | Monte Carlo sampling |
| | | `'LHS'` | Latin Hypercube sampling |
| | | `'Sobol'` | Sobol sequence sampling |
| | | `'Halton'` | Halton sequence sampling |
| ☐ | `.Nsamples` | Integer | The number of samples to draw. It is required when `.Sampling` is specified. |
| ⊕ | `.X` | $N \times M$ Double | User defined experimental design X. If specified, `.Sampling` is ignored. |
| ⊕ | `.Y` | $N \times N_{Out}$ Double | User defined model response Y. If specified, `.Sampling` is ignored. |
| ⊕ | `.DataFile` | String | `mat`-file containing the experimental design. If specified, `.Sampling` is ignored. |

## 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 10.

| Table 10: `myPCE` | | |
|---|---|---|
| `.Name` | String | Unique name of the PCE metamodel |
| `.Options` | Table 3 | Copy of the `MetaOpts` structure used to create the metamodel |
| `.PCE` | Table 11 | Information about all the elements of Eq. (1.3) |
| `.ExpDesign` | Table 14 | Experimental design used for calculating the coefficients |
| `.Error` | Table 15 | Error measures of the metamodelling calculation results (Section 1.4.3) |
| `.Internal` | Table 16 | 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 11.

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 11: `myPCE.PCE(i)` | | |
|---|---|---|
| `.Coefficients` | $P \times 1$ Double | Truncated PCE coefficients. |
| `.Moments` | Table 12 | Post-processed moments of the PCE (Section 1.6). |
| `.Basis` | Table 13 | Information about the truncated polynomial basis. |

| Table 12: `myPCE.PCE(i).Moments` | | |
|---|---|---|
| `.Mean` | Double | Mean of the PCE (Eq. (1.36)) |
| `.Var` | Double | Variance of the PCE (Eq. (1.37)) |

| Table 13: `myPCE.PCE(i).Basis` | | |
|---|---|---|
| `.Degree` | Double | Maximum 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 |

### 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 14: `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 1.3.3) |
| `.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 two error estimates described in 1.4.3.3 are available in the `myPCE.Error` output field, as described in Table 15.

| Table 15: `myPCE.Error` | | |
|---|---|---|
| `.LOO` | Double | Leave-One-Out error (see section 1.4.3). |
| `.normEmpError` | Double | Normalized Empirical Error (see section 1.4.3) |

### 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 16. Note that not all the fields are always available, as they depend on the original configuration options.

Table 16: `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 17 | Additional information about the PCE error estimation given in `myPCE.Error` |
| `.PCE` | Table 18 | Additional information on the PCE calculation. |
| `.Runtime` | Table 23 | Temporary variables and configuration flags used during the calculation of the PCE coefficients |

Table 17: `myPCE.Internal.Error`

| | | |
|---|---|---|
| `.LOO_lars` | Double | LOO error as calculated by LARS (before hybrid LARS) |
| `.LOO_omp` | Double | LOO error as calculated by OMP |

Table 18: `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 19 | OLS-specific information |
| `.LARS` | Table 20 | LARS-specific information |
| `.OMP` | Table 21 | OMP-specific information |
| `.Basis` | Table 22 | Miscellaneous information about the polynomial basis (e.g., truncation parameters) |

| Table 19: 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. |
| .normEmpError | Array | Normalized empirical error for each of the tested degrees in degree-adaptive mode. |

| Table 20: myPCE.Internal.PCE.LARS | | |
|---|---|---|
| .TargetAccuracy | Double | Degree-adaptive early stop threshold |
| .LarsEarlyStop | Logical | Early stop in LARS iterations flag |
| .HybridLars | Logical | Enable/disable hybrid lars |
| .ModifiedLoo | Logical | Enable/Disable the “modified LOO” error estimation in Eq. (1.27) |
| .LOO | Array | LOO error for each of the tested degrees in degree-adaptive mode. |
| .normEmpError | 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 21: 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. (1.27) |
| .LOO | Vector of doubles | LOO error for each of the tested degrees in degree-adaptive mode. |
| .normEmpError | 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 22: myPCE.Internal.PCE.Basis | | |
|---|---|---|
| .Truncation | Structure | Structure with the truncation options used to generate the basis. See Table 4 |

| Table 23: myPCE.Internal.Runtime | | |
|---|---|---|
| .isInitialized | Logical | A flag that determines whether the current meta-model 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 meta-model have been calculated |
| .Nout | Integer | The Output dimension |
| .current_output | Integer | The current output (This is used during the calculation of the meta-model) |
| .degree_index | Integer | Index of the PCE being considered in the current status of the calculation |

# References

Berveiller, M., B. Sudret, and M. Lemaire (2006). Stochastic finite elements: a non intrusive approach by regression. *Eur. J. Comput. Mech. 15*(1-3), 81–92. 7

Blatman, G. (2009). *Adaptive sparse polynomial chaos expansion for uncertainty propagation and sensitivity analysis*. Ph. D. thesis, Blaise Pascal University - Clermont II. 4, 9

Blatman, G. and B. Sudret (2010). An adaptive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis. *Prob. Eng. Mech. 25*(2), 183–197. 9

Blatman, G. and B. Sudret (2011). Adaptive sparse polynomial chaos expansion based on Least Angle Regression. *J. Comput. Phys. 230*, 2345–2367. 10, 11

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

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

Gander, W. and W. Gautschi (2000). Adaptive quadrature revisited. *BIT Numerical Mathematics 40*(1), 84–101. 6

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. 3, 6

Gerstner, T. and M. Griebel (1998). Numerical integration using sparse grids. *Numer. Algorithms 18*(3-4), 209–232. 6

Golub, G. H. and J. H. Welsch (1969). Calculation of gauss quadrature rules. *Mathematics of computation 23*(106), 221–230. 6

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

Ishigami, T. and T. Homma (1990). An importance quantification technique in uncertainty analysis for computer models. In *Proc. ISUMA'90, First Int. Symp. Unc. Mod. An*, pp. 398–403. University of Maryland. 17

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

Migliorati, G., F. Nobile, E. Von Schwerin, and R. Tempone (2013). Approximation of quantities of interest in stochastic PDEs by the random discrete L2 projection on polynomial spaces. *SIAM J. Sci. Comput. 35*(3), 1440–1460. 8

Pati, Y., R. Rezaiifar, and P. Krishnaprasad (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*, pp. 40–44. 13

Shampine, L. (2008). Vectorized adaptive quadrature in MATLAB. *Journal of Computational and Applied Mathematics 211*(2), 131–140. 3

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. *J. Royal Stat. Soc., Series B 58*, 267–288. 11

Xiu, D. and G. E. Karniadakis (2002). The Wiener-Askey polynomial chaos for stochastic differential equations. *SIAM J. Sci. Comput. 24*(2), 619–644. 3