ETH zürich

UQLab User Manual
Generalized Lambda Models (GLaM)

N. Lüthen, X. Zhu, 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

N. Lüthen, X. Zhu, S. Marelli, B. Sudret, UQLab user manual – Generalized Lambda Models (GLaM), Report UQLab-V2.1-120, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland, 2024

BibTeX entry

```
@TechReport{UQdoc_21_120,
author = {Lüthen, N. and Zhu, X. and Marelli, S. and Sudret, B.},
title = {{UQLab user manual -- Generalized Lambda Model}},
institution = {Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich,
Switzerland},
year = {2024},
note = {Report UQLab-V2.1-120}
}
```

Document Data Sheet

Document Ref.	UQLAB-V2.1-120
Title:	UQLAB user manual – Generalized Lambda Models (GLaM)
Authors:	N. Lüthen, X. Zhu, S. Marelli, B. Sudret Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland
Date:	15/04/2024

Doc. Version	Date	Comments
V2.1	15/04/2024	UQLAB V2.1 release (first version)

Abstract

The generalized lambda model (GLaM) is a novel metamodelling technique for stochastic simulators, *i.e.* for computational models whose response to a given set of input parameters is not a deterministic value, but a realization of a random variable with unknown probability density function (PDF). GLaM is a surrogate designed to accurately predict this response PDF for any set of input parameters by approximating it with the flexible family of generalized lambda distributions. The associated distribution parameters are cast as deterministic functions of the input parameters, and represented by polynomial chaos expansions. The surrogate can be built from experimental designs comprising replicated model evaluations as well as from replication-free experimental designs.

The UQLAB GLaM module offers an easy way to construct GLaM surrogate models based on replicated and replication-free experimental designs. It makes use of other UQLAB modules (UQLAB User Manual – the INPUT module, UQLAB User Manual – the MODEL module) to define the input distribution and the stochastic model. It also relies on the UQLAB PCE-module (UQLAB User Manual – Polynomial Chaos Expansions) for approximating the distribution parameter functions. The manual for the GLaM metamodelling module is divided into three parts:

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

Keywords: UQLAB, metamodelling, stochastic model, generalized lambda distribution, polynomial chaos expansion

Contents

1 Theory **1**

1.1 Introduction . . . 1

1.2 Basics . . . 1

1.2.1 Stochastic simulators . . . 1

1.2.2 Generalized lambda distribution . . . 2

1.2.3 Polynomial chaos expansions . . . 3

1.2.4 Fitting GLDs to data . . . 4

1.3 Generalized lambda models (GLaMs) . . . 5

1.3.1 Framework . . . 5

1.3.2 Likelihood function . . . 5

1.3.3 Optimization . . . 6

1.3.4 Joint PCE-GLD fitting based on a design with replications . . . 6

1.3.5 Fitting the GLaM by full regression without replications . . . 7

2 Usage **9**

2.1 Reference problem: geometric Brownian motion . . . 9

2.2 Problem setup . . . 9

2.2.1 Full model and probabilistic input model . . . 9

2.3 Setup of a GLaM metamodel . . . 10

2.3.1 Fitting a GLaM with replications (RepJoint) . . . 10

2.3.2 Fitting a GLam without replications (FullReg) . . . 11

2.4 Accessing the results . . . 11

2.4.1 PCEs modelling the lambda functions . . . 13

2.4.2 Error . . . 13

2.4.3 Experimental design . . . 13

2.5 Advanced options . . . 13

2.5.1 User-defined experimental design . . . 13

2.5.2 Truncation options . . . 14

2.5.3 Use of a validation set . . . 14

2.5.4 Vector-valued models . . . 14

2.5.5 Method-specific options . . . 15

2.5.6 Using GLaM as a model (predictor) . . . 16

2.5.7 Custom GLaM 16

3 Reference List **19**

3.1 Create a GLaM metamodel 21

3.1.1 Lambda options 22

3.1.2 Method-specific options 23

3.1.3 Experimental design 24

3.1.4 Validation Set 24

3.2 Accessing the results 24

3.3 Additional functions 26

3.3.1 `uq_display` 26

3.3.2 `uq_evalStochSimMetrics` 27

3.3.3 `uq_GLaM_evalLambda` 28

3.3.4 `uq_GLD_DistributionFunc` 29

Chapter 1

Theory

1.1 Introduction

In uncertainty quantification, two classes of models can be distinguished. For *deterministic models*, the uncertainty in the output originates exclusively from the uncertainty in the input variables. If the values of the inputs are held fixed, the output is fixed, too. On the other hand, there are models whose output is random, even if all input variables are set to fixed values. In other words, for each vector of input parameters, a model evaluation is a realization from an associated random variable. Such models are called *stochastic simulators*.
The generalized lambda model (GLaM) is a surrogate model developed for the case of stochastic simulators (Zhu and Sudret, 2020, 2021a,b; Zhu et al., 2022). Its aim is to represent the probability density function (PDF) of the random variable response for any vector of input parameters. It uses the flexible distribution family *generalized lambda distribution* to represent the PDF of the model response conditioned on the input parameters. To achieve this, the four parameters of the generalized lambda distribution are approximated as a function of the model inputs by polynomial chaos expansions (PCE). The coefficients and parameters of the PCE can be determined by different optimization methods.
In Section 1.2 we review the components of the method, before we describe GLaM in detail in Section 1.3.

1.2 Basics

1.2.1 Stochastic simulators

A computational model can be seen as a black box, *i.e.* as a map from the space of input parameters to that of output quantities:

$$\boldsymbol{y} = \mathcal{M}(\boldsymbol{x}) \tag{1.1}$$

where $\boldsymbol{x}$ is a realization of the random vector $\boldsymbol{X}$ that parametrizes the variability of the input parameters (typically through a joint probability density function (PDF)), and $\boldsymbol{y}$ is the vector of model responses.

We identify two classes of models:

- *deterministic* models, which will always give the same result $\boldsymbol{y}_0 = \mathcal{M}(\boldsymbol{x}_0)$ when evaluated repeatedly for a given input realization $\boldsymbol{x}_0$.
- *stochastic* models, which for a given input realization $\boldsymbol{x}_0$ return an output random variable $\boldsymbol{Y}_0 = \mathcal{M}(\boldsymbol{x}_0)$ with distribution $f_{\boldsymbol{Y}_0}$. Every time a stochastic model is evaluated, a realization $\boldsymbol{y}_0 \sim \boldsymbol{Y}_0$ is generated. A model from the this class is also called *stochastic simulator*.

Stochastic models can also be interpreted as a function from the input space to the set of conditional response PDFs, mapping an input vector to the PDF of the corresponding response random variable, or more precisely, as a conditional sampler with the underlying probability distribution given by

$$f_{\mathcal{M}(\cdot)} : \boldsymbol{x}_0 \mapsto f_{\mathcal{M}(\boldsymbol{x}_0)}.$$

For a detailed description of how stochastic simulators are treated and can be implemented in UQLAB, the reader is referred to the UQLAB User Manual – the MODEL module.

Replications

For stochastic simulators, the output $\boldsymbol{Y}_0 = \mathcal{M}(\boldsymbol{x}_0)$ is a random variable. Evaluating the model several times for the same input vector $\boldsymbol{x}_0$ generates several independent realizations $\boldsymbol{y}_0^{(1)}, \ldots, \boldsymbol{y}_0^{(n)} \sim \boldsymbol{Y}_0$, called *replications*.
Replications can be used to characterize the PDF $f_{\boldsymbol{Y}_0}$ of the response random variable at $\boldsymbol{x}_0$.
A set of model evaluations that contains replications for all points of the experimental design is called *replication-based*.
Conversely, we call a dataset that contains only a single realization of the response random for each input realization *replication-free*.

1.2.2 Generalized lambda distribution

The generalized lambda distribution (GLD) is a flexible probability distribution family commonly used to approximate the behavior of other well-known parametric distributions, such as uniform, normal, Weibull, and Student's t (Freimer et al., 1988). GLDs can produce a wide range of shapes, including bell-shaped, U-shaped, S-shaped and bounded-mode distributions, as long as they are unimodal. A graphical depiction of the flexibiliy of GLDs can be found in Figure 1.
The definition of a GLD relies on a parametrization of the *quantile function* $Q(u)$, which is a nondecreasing function defined on $[0, 1]$. The GLaM model uses the GLD of the Freimer–Kollia–Mudholkar–Lin family (Freimer et al., 1988), which is defined by

$$Q(u; \boldsymbol{\lambda}) = \lambda_1 + \frac{1}{\lambda_2}\left(\frac{u^{\lambda_3} - 1}{\lambda_3} - \frac{(1-u)^{\lambda_4} - 1}{\lambda_4}\right), \tag{1.2}$$

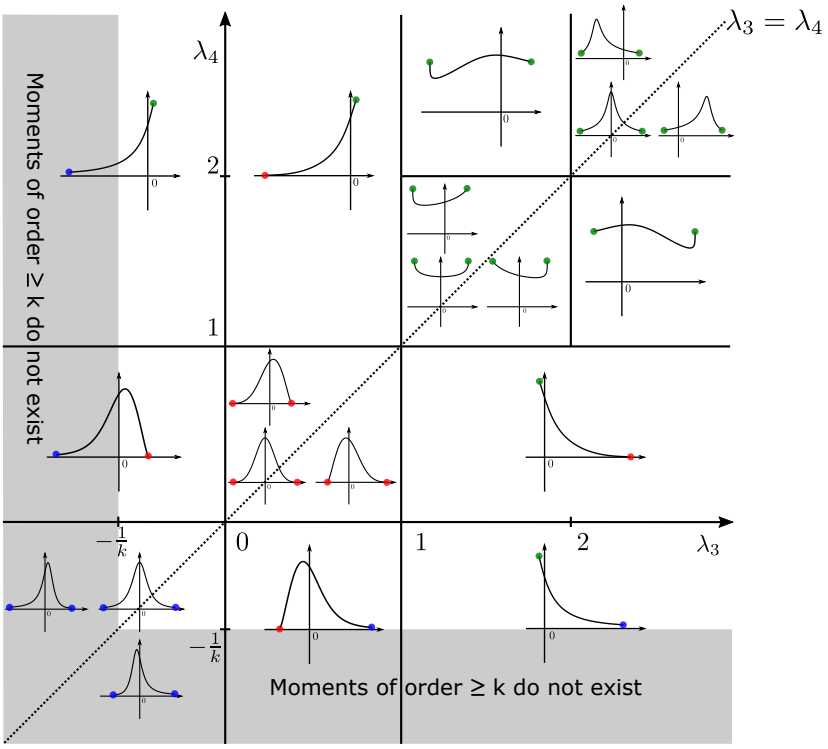

Figure 1: Graphical depiction of the distribution shapes that can be approximated by the family of FKML generalized lambda distributions, as a function of its shape parameters λ_3 and λ_4. From Zhu and Sudret (2021a).

where $\boldsymbol{\lambda} = \{\lambda_l : l = 1, \ldots, 4\}$ are the four distribution parameters. Here, λ_1 is the location parameter, λ_2 is the scaling parameter (required to be positive), and λ_3 and λ_4 are the shape parameters. Based on the quantile function, the PDF $f_W(w; \boldsymbol{\lambda})$ of a random variable W following a GLD can be derived as

$$f_W(w; \boldsymbol{\lambda}) = \frac{1}{Q'(u; \boldsymbol{\lambda})} = \frac{\lambda_2}{u^{\lambda_3 - 1} + (1-u)^{\lambda_4 - 1}} \mathbb{1}_{[0,1]}(u), \text{ with } u = Q^{-1}(w; \boldsymbol{\lambda}), \tag{1.3}$$

where $Q'(u; \boldsymbol{\lambda})$ is the derivative of Q with respect to u, and $\mathbb{1}_{[0,1]}$ is the indicator function. A closed-form expression of Q^{-1}, and therefore of f_W, is in general not available, and thus the PDF is evaluated by solving the nonlinear Eq. (1.3) numerically.

1.2.3 Polynomial chaos expansions

Consider a random vector with independent components $\boldsymbol{X} \in \mathbb{R}^M$ described by the joint probability density function (PDF) $f_{\boldsymbol{X}}$. Polynomial Chaos Expansions (PCE) approximates the computational model output $Y = \mathcal{M}(\boldsymbol{X})$ by a sum of orthonormal polynomials (Xiu and Karniadakis, 2002; Sudret, 2007):

$$Y \approx \mathcal{M}^{\text{PC}} = \sum_{\boldsymbol{\alpha} \in \mathcal{A}} c_{\boldsymbol{\alpha}} \Psi_{\boldsymbol{\alpha}}(\boldsymbol{X}), \tag{1.4}$$

where $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X})$ are multivariate polynomials orthonormal with respect to the input distribution $f_{\boldsymbol{X}}$, $\boldsymbol{\alpha} \in \mathcal{A} \subset \mathbb{N}^M$ are multi-indices, and $c_{\boldsymbol{\alpha}}$ are the corresponding coefficients. For more details, the reader is referred to UQLAB User Manual – Polynomial Chaos Expansions.

1.2.4 Fitting GLDs to data

To fit a GLD $f(y;\boldsymbol{\lambda})$ to given data $\mathcal{Y} = \left\{y^{(1)}, \ldots, y^{(N)}\right\}$, two popular methods are the *method of moments* and the *maximum likelihood method*, which are briefly described below. Other methods are reviewed by Chalabi (2012).

1.2.4.1 Method of moments

The method of moments relies on the idea that the moments of the data set and the fitted distribution should coincide. For a distribution family with n parameters, the first n moments are matched to the empirical moments computed from the data. For the four-parameter family of GLD, the mean, variance, skewness, and kurtosis are computed analytically in terms of the parameters $\lambda_1, \ldots, \lambda_4$:

$$\mu = \mathbb{E}\left[Y\right] = \lambda_1 - \frac{1}{\lambda_2}\left(\frac{1}{\lambda_3+1} - \frac{1}{\lambda_4+1}\right), \tag{1.5}$$

$$\sigma^2 = \mathbb{E}\left[(Y-\mu)^2\right] = \frac{(v_2 - v_1^2)}{\lambda_2^2}, \tag{1.6}$$

$$\delta = \mathbb{E}\left[\left(\frac{(Y-\mu)}{\sigma}\right)^3\right] = \frac{v_3 - 3v_1v_2 + 2v_1^3}{(v_2 - v_1^2)^{\frac{3}{2}}}, \tag{1.7}$$

$$\kappa = \mathbb{E}\left[\left(\frac{(Y-\mu)}{\sigma}\right)^4\right] = \frac{v_4 - 4v_1v_3 + 6v_1^2v_2 - 3v_1^4}{(v_2 - v_1)^2}, \tag{1.8}$$

where v_k is an auxiliary function defined by

$$v_k = \int_0^1 s(u)^k du = \sum_{j=0}^{k} \frac{(-1)^j}{\lambda_3^{k-j}\lambda_4^j}\binom{k}{j} B(\lambda_3(k-j)+1, \lambda_4 j + 1). \tag{1.9}$$

Here, B denotes the beta function. After matching these formulas to the corresponding empirical values, the equations are solved for $\lambda_1, \ldots, \lambda_4$.

1.2.4.2 Maximum likelihood

The maximum likelihood method determines the distribution parameters by maximizing the probability that the data was generated by a distribution with these parameters. In practice, the negative logarithm of the likelihood is minimized, *i.e.*

$$\hat{\boldsymbol{\lambda}} = \arg\min_{\boldsymbol{\lambda}} l(\boldsymbol{\lambda}) \tag{1.10}$$

$$\text{where } l(\boldsymbol{\lambda}) = -\sum_{i=1}^{N} \log\left(f(y^{(i)};\boldsymbol{\lambda})\right) \tag{1.11}$$

Since the PDF of GLD is not given explicitly, each evaluation of 1.11 requires N solutions of the nonlinear equation $u = Q^{-1}(y; \boldsymbol{\lambda})$ (see Eq. (1.3)).

1.3 Generalized lambda models (GLaMs)

1.3.1 Framework

Generalized lambda models, or GLaMs in short, are a class of surrogate models designed to reproduce the behavior of stochastic simulators, originally proposed in Zhu and Sudret (2020, 2021a,b). The GLaM *stochastic emulator* assumes that each response random variable $Y_{\boldsymbol{x}_0} = \mathcal{M}(\boldsymbol{x}_0)$ follows a GLD. Hence, the distribution parameters $\boldsymbol{\lambda}$ are functions of the input variables:

$$Y(\boldsymbol{x}) \sim \mathrm{GLD}\left(\lambda_1(\boldsymbol{x}), \lambda_2(\boldsymbol{x}), \lambda_3(\boldsymbol{x}), \lambda_4(\boldsymbol{x})\right). \tag{1.12}$$

Each component of $\boldsymbol{\lambda}(\boldsymbol{x})$ is represented by a PCE (see 1.2.3). Since $\lambda_2(\boldsymbol{x})$ must be positive, the associated PCE is built on the natural logarithm transform $\log\left(\lambda_2(\boldsymbol{x})\right)$. This results in the following approximations:

$$\lambda_l\left(\boldsymbol{x}\right) \approx \lambda_l^{\mathrm{PC}}\left(\boldsymbol{x}; \boldsymbol{c}\right) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}_l} c_{l,\boldsymbol{\alpha}} \psi_{\boldsymbol{\alpha}}(\boldsymbol{x}), \quad l = 1, 3, 4, \tag{1.13}$$

$$\lambda_2\left(\boldsymbol{x}\right) \approx \lambda_2^{\mathrm{PC}}\left(\boldsymbol{x}; \boldsymbol{c}\right) = \exp\left(\sum_{\boldsymbol{\alpha} \in \mathcal{A}_2} c_{2,\boldsymbol{\alpha}} \psi_{\boldsymbol{\alpha}}(\boldsymbol{x})\right), \tag{1.14}$$

where $\boldsymbol{\mathcal{A}} = \{\mathcal{A}_l : l = 1, \ldots, 4\}$ are the truncation sets defining the basis functions (see also UQLab User Manual – Polynomial Chaos Expansions), and $\boldsymbol{c} = \{c_{l,\boldsymbol{\alpha}} : l = 1, \ldots, 4,\ \boldsymbol{\alpha} \in \mathcal{A}_l\}$ are coefficients associated to the polynomial basis.
Constructing a GLaM surrogate consists in determining the truncation sets $\boldsymbol{\mathcal{A}}$ and the PCE coefficients $\boldsymbol{c}$. The basic idea of GLaM is to determine $\boldsymbol{c}$ using the maximum likelihood method, as explained in the following paragraph.

1.3.2 Likelihood function

For an experimental design $\mathcal{X} = \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\right\}$ with associated model responses $\mathcal{Y} = \left\{y^{(1)}, \ldots, y^{(N)}\right\}$, define an optimization problem in terms of the parameters $\boldsymbol{c}$ of the surrogate:

$$\hat{\boldsymbol{c}} = \arg\max_{\boldsymbol{c} \in \mathcal{C}} \mathsf{L}\left(\boldsymbol{c}\right), \tag{1.15}$$

where the objective function is the maximum conditional likelihood estimator

$$\mathsf{L}\left(\boldsymbol{c}\right) = \sum_{i=1}^{N} \log\left(f\left(y^{(i)}; \boldsymbol{\lambda}^{\mathrm{PC}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{c}\right)\right)\right). \tag{1.16}$$

Here, f denotes the PDF of the GLD defined in Eq. (1.3). $\boldsymbol{\lambda}^{\mathrm{PC}}$ is the vector-valued function of GLD parameters, defined over the input space $\mathcal{D}_{\boldsymbol{X}}$ and parametrized by the set of PCE coefficients $\boldsymbol{c}$. $\mathcal{C}$ is the search space for $\boldsymbol{c}$. The objective function in Eq. (1.16) is obtained by

minimizing the Kullback-Leibler divergence between the surrogate PDF and the underlying true response PDF over $\mathcal{D}_{\boldsymbol{X}}$ (Zhu and Sudret, 2020, 2021a).

1.3.3 Optimization

The optimization problem Eq. (1.15) is challenging to solve because the PDF of GLDs does not have an explicit form (Eq. (1.3)), because Eq. (1.16) is highly nonlinear, and because Eq. (1.15) is subject to complex inequality constraints due to the dependence of the PDF support on $\boldsymbol{\lambda}$. For more insight into this complex interdependence, and how it affects the fitting procedure, the reader is directed to Zhu and Sudret (2021a).

Given a starting point, first the *derivative-based trust-region* optimization algorithm is applied without constraints. If none of the inequality constraints are activated at the optimum, the results are kept as the final estimates. Otherwise, the constrained (1+1)-CMA-ES algorithm, available in UQLIB user manual, is used instead.

The GLaM module provides two methods for fitting a GLaM model to given data. The method `'RepJoint'`, introduced by Zhu and Sudret (2020), requires replications. It is explained in Section 1.3.4. The method `'FullReg'`, introduced by Zhu and Sudret (2021a), can be used both with and without replications. It is explained in Section 1.3.5. The methods differ mainly in how they determine the basis functions $\Psi_{\boldsymbol{\alpha}}$ to be included in the PCEs of $\lambda_1, \ldots, \lambda_4$, which in turn defines which of the coefficients $c_{\boldsymbol{\alpha}}$ are part of the optimization procedure in Eq. (1.15).

1.3.4 Joint PCE-GLD fitting based on a design with replications

Consider an experimental design with replications $\mathcal{X} = \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\right\}$, typically a sample from the input distribution $f_{\boldsymbol{X}}(\boldsymbol{x})$, and a corresponding replication-based set of model evaluations $\mathcal{Y} = \left\{y^{(1,1)}, \ldots, y^{(1,R)}, y^{(2,1)}, \ldots, y^{(2,R)}, \ldots, y^{(N,1)}, \ldots, y^{(N,R)}\right\}$.

The joint fitting method `'RepJoint'` consists of the following steps:

1. For each $i = 1, \ldots, N$, a GLD is fitted to the samples $y^{(i,1)}, \ldots, y^{(i,R)}$. This results in a set of four parameter estimates $\left(\hat{\lambda}_1^{(i)}, \ldots, \hat{\lambda}_4^{(i)}\right)$ associated to each $\boldsymbol{x}^{(i)}$.

2. The parameter estimates are treated as samples from the unknown vector-valued function $\boldsymbol{\lambda}(\boldsymbol{x})$. A regression method (e.g., LARS) is used to fit a PCE with four outputs to this data.

3. Finally, the coefficients for the selected basis are recomputed by jointly optimizing the likelihood function defined in Eq. (1.16). The optimization procedure is explained in Section 1.3.3.

For more details, the reader is referred to Zhu and Sudret (2020).

1.3.5 Fitting the GLaM by full regression without replications

Let an experimental design $\mathcal{X}, \mathcal{Y}$ with or without replications be given. Instead of the sequential approach described in the previous section, the full regression fitting method `'FullReg'` aims at directly minimizing Eq. (1.16) and therefore does not require replications.
The full regression fitting method `'FullReg'` consists of the following steps:

1. Because $\mathsf{L}(\boldsymbol{c})$ is highly nonlinear, a good starting point is necessary to guarantee the convergence of the optimization algorithm. The starting point is determined as follows. Under the assumption that $\lambda_3(\boldsymbol{x})$ and $\lambda_4(\boldsymbol{x})$ do not vary strongly over the domain $\mathcal{D}_{\boldsymbol{X}}$, analytical calculations show that the variations of the mean function $\mu(\boldsymbol{x})$ and the variance function $v(\boldsymbol{x})$ of the GLaM are dominated by the location parameter $\lambda_1(\boldsymbol{x})$ and the scale parameter $\lambda_2(\boldsymbol{x})$, respectively (Zhu and Sudret, 2021a), see also Eqs. (1.5) and (1.6). Following this property, feasible generalized least squares (FGLS) (Wooldridge, 2013) is used to jointly fit PCEs to $\mu(\boldsymbol{x})$ and $\log\left(v(\boldsymbol{x})\right)$ in an iterative manner. Degree- and q-norm adaptivity are used to identify the best truncation scheme for the two PCEs.

2. After obtaining coefficient vectors for $\mu(\boldsymbol{x})$ and $\log\left(v(\boldsymbol{x})\right)$ (or $\lambda_1(\boldsymbol{x})$ and $\log\left(\lambda_2(\boldsymbol{x})\right)$, respectively) from FGLS, two rounds of the optimization procedure described in Section 1.3.3 are performed to build the full GLaM surrogate. In both rounds, the PCEs for $\lambda_1(\boldsymbol{x})$ and $\lambda_2(\boldsymbol{x})$ reuse the basis functions identified during Step 1.

   - In the first round, the PCEs for $\lambda_3(\boldsymbol{x})$ and $\lambda_4(\boldsymbol{x})$ are of order 0 (i.e, constant) and thus contribute only one free parameter each. The coefficient vectors of all four parameter functions are recomputed. The starting point is the solution from Step 1, together with values for the constant coefficients of $\lambda_3(\boldsymbol{x})$ and $\lambda_4(\boldsymbol{x})$ that correspond to the shape of a Gaussian distribution.
   - In the second round, the PCEs for $\lambda_3(\boldsymbol{x})$ and $\lambda_4(\boldsymbol{x})$ also include non-constant terms. Again, all coefficient vectors are recomputed. The starting point is the optimum from the previous round.

Remark: Since a GLD can have very fat tails, solving the optimization problem may produce response PDFs with unexpected infinite moments, especially when the model is trained on a small data set. To prevent excessively fat tails, we apply the threshold $\lambda_3^{\mathrm{PC}}(\boldsymbol{x}) = \max\left\{\lambda_3^{\mathrm{PC}}(\boldsymbol{x};\hat{\boldsymbol{c}}), -0.2\right\}$ and $\lambda_4^{\mathrm{PC}}(\boldsymbol{x}) = \max\left\{\lambda_4^{\mathrm{PC}}(\boldsymbol{x};\hat{\boldsymbol{c}}), -0.2\right\}$, which indicates that we enforce the surrogate PDFs to have finite moments up to order 4. When enough data are available, these operations are unnecessary because the resulting model does not exceed the threshold.
For more details, inlcuding an in-depth description of the degree- and q-norm adaptive algorithm adopted for λ_3 and λ_4, the reader is referred to Zhu et al. (2022).

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: geometric Brownian motion

A classical example of stochastic simulator is the geometric Brownian motion model (GBM), which is defined by the stochastic differential equation:

$$dS_t(\boldsymbol{x}) = x_1 S_t + x_2 \, dW_t, \tag{2.1}$$

with the boundary condition $S_0(\boldsymbol{x}) = 1$. Here, W_t is a Wiener process, and x_1 and x_2 are parameters of the equation. The model we consider simulates the value of a GBM at time 1, *i.e.* $Y_{\boldsymbol{x}} = S_1(\boldsymbol{x})$. Analytical calculations show that for every $\boldsymbol{x}$, the model response $Y_{\boldsymbol{x}}$ follows a lognormal distribution

$$Y_{\boldsymbol{x}} \sim \mathcal{LN}(x_1 - x_2^2/2, x_2). \tag{2.2}$$

This model is available in UQLAB under the name `Y = uq_GBM(X)`.
In this example, the two parameters x_1 and x_2 are considered random variables $X_1 \sim \mathcal{U}(0, 0.1), X_2 \sim \mathcal{U}(0.1, 0.4)$.

2.2 Problem setup

The UQLAB framework is first initialized with the following command:

```
uqlab
```

2.2.1 Full model and probabilistic input model

To surrogate the GBM model using UQLAB, we need to first define a basic stochastic MODEL object:

```
ModelOpts.mFile = 'uq_GBM';
ModelOpts.isStochastic = true;
```

```
ModelOpts.isVectorized = true;
myModel = uq_createModel(modelopts);
```

Setting the flag `isStochastic` to `true` causes UQLAB to treat this model as a stochastic simulator.

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

Due to the internal use of PCE, a GLaM model *always* requires an input model, even when providing a custom experimental design. The INPUT object is defined by:

```
InputOpts.Marginals(1).Type = 'Uniform';
InputOpts.Marginals(1).Parameters = [0, 0.1];
InputOpts.Marginals(2).Type = 'Uniform';
InputOpts.Marginals(2).Parameters = [0.1, 0.4];

myInput = uq_createInput(InputOpt);
```

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 a GLaM metamodel

The GLAM module creates a MODEL object that can be used as any other stochastic model. The basic options common to any GLaM surrogate MODEL read:

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

Regardless of how the parameters of the PCE models are determined (see Section 1.3.4 and Section 1.3.5), the adaptive degree for the PCE models of each of the $\lambda_1, \ldots, \lambda_4$ is by default set to `0:3` for λ_1, `0:2` for λ_2, and `0:1` for the remaining parameters. Nevertheless, they can be manually set to the desired values by adding the `MetaOpts.Lambda(i)` options:

```
  MetaOpts.Lambda(1).Degree=0:3;
  MetaOpts.Lambda(2).Degree=0:2;
  MetaOpts.Lambda(3).Degree=0:1;
  MetaOpts.Lambda(4).Degree=0:1;
```

The additional configuration options needed to properly create a GLaM object in UQLAB are given in the following subsections.

2.3.1 Fitting a GLaM with replications (RepJoint)

Specify the replication-based fitting method (*i.e.* maximum likelihood estimation with full basis selection for λ_3 and λ_4):

```
MetaOpts.Method = 'RepJoint';
```

This method requires replications. To generate an experimental design of size 30 based on the latin hypercube sampling, with 25 replications in each point, the following commands

are used:

```
MetaOpts.ExpDesign.Sampling = 'LHS';
MetaOpts.ExpDesign.NSamples = 30;
MetaOpts.ExpDesign.Replications = 25;
```

The GLaM is created by:

```
myGLaM = uq_createModel(MetaOpts);
```

2.3.2 Fitting a GLam without replications (FullReg)

The fitting method `'FullReg'` (*i.e.* maximum likelihood estimation with full basis selection for λ_3 and λ_4) is specified as follows:

```
MetaOpts.Method = 'FullReg';
```

This fitting method does not require the experimental design to feature replications. To create a replication-free experimental design of size 800 based on latin hypercube sampling, use the following options:

```
MetaOpts.ExpDesign.Sampling = 'LHS';
MetaOpts.ExpDesign.NSamples = 750;
```

Then the GLaM is created by:

```
myGLaM = uq_createModel(MetaOpts);
```

2.4 Accessing the results

Once the model is created, a report with basic information about the GLaM metamodel can be printed as follows (here shown for the model created in Section 2.3.2):

```
uq_print(myGLaM)
```

which produces the following output in the MATLAB command window:

```
%------------ Generalized lambda model output ---------------%
   Number of input variables:                  2
   Full model evaluations:                     750 ED points with 1 replications
   AIC:                                        -4.9113568e+01
   BIC:                                        -2.9128361e+00

 **Lambda1:
   Maximal degree:                             3
   Size of full basis:                         10
   Size of sparse basis:                       3


 **Lambda2:
   Maximal degree:                             1
   Size of full basis:                         3
   Size of sparse basis:                       3
```

```
**Lambda3:
  Maximal degree:                          1
  Size of full basis:                      3
  Size of sparse basis:                    3


**Lambda4:
  Maximal degree:                          0
  Size of full basis:                      1
  Size of sparse basis:                    1

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

Similarly, a visual representation of the GLaM can be obtained as follows:

```
uq_display(myGLaM)
```

which plots the median and the 2.5%-97.5% quantile (for 1-dimensional inputs) or the mean and variance (for 2-dimensional inputs) of the GLaM predictor, as shown in Figure 2.

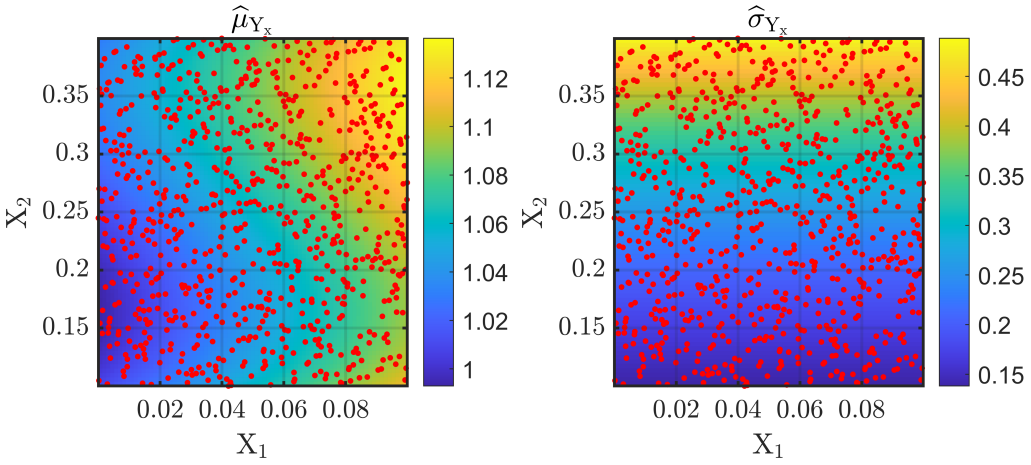

Figure 2: The figure created by `uq_display` of a GLaM MODEL object which has a two-dimensional input. On the left is the mean function and on the right the standard deviation function in the input space. The red dots represent the experimental design.

More options for `uq_display` as well as three additional useful postprocessing and display functions are listed in Section 3.3.

Further results are stored in the object `myGLaM` that was created by UQLAB. We can directly access this object in order to obtain various results and information on the metamodel:

```
>> myGLaM
  uq_model with properties:

        Internal: [1x1 struct]
            Name: 'Model 2'
            Type: 'uq_metamodel'
       ExpDesign: [1x1 struct]
            GLaM: [1x4 struct]
```

```
    isStochastic: 1
         Options: [1x1 struct]
           Error: [1x1 struct]
```

A complete list of results is provided in Section 3.2.

2.4.1 PCEs modelling the lambda functions

The field `myGLaM.GLaM` contains a 1×4 struct array with all information about the PCEs modeling the parameter functions $\lambda_1(\boldsymbol{x}), \ldots, \lambda_4(\boldsymbol{x})$. This includes their basis and the associated coefficients (see also UQLAB User Manual – Polynomial Chaos Expansions).

2.4.2 Error

To assess the goodness-of-fit of the calculated metamodel, UQLAB provides several error measures. These include the negative log-likelihood, *i.e.* the value of Eq. (1.16) after optimization, as well as various model selection criteria that also take the number of free parameters into account. For a review of these model selection criteria, the reader is referred to Ye et al. (2008).
If a validation set is provided (see Section 2.5.3), the field `.Val` contains several validation error metrics as explained in Section 3.3.2.

2.4.3 Experimental design

The field `myGLaM.ExpDesign` contains all information about the experimental design, whether it was provided by the user or sampled by UQLAB. This includes information such as the number of samples, the number of replications, and the input samples as well as the corresponding model evaluations.

2.5 Advanced options

2.5.1 User-defined experimental design

Instead of letting UQLAB create an experimental design, it is also possible to provide a data set using the following syntax:

```
MetaOpts.ExpDesign.X = X_data;
MetaOpts.ExpDesign.Y = Y_data; % corresponding model evaluations
```

If an experimental design is provided in this way, the fields `.Sampling`, `.NSamples`, and `Replications` are ignored.
The format of `Y_data` follows the convention of stochastic simulators introduced in the UQLAB User Manual – the MODEL module, *i.e.* a collection of N model responses with N_{Out} model outputs and N_R replications must be provided as a 3-dimensional matrix of dimensions $N \times N_{\text{Out}} \times N_R$.

Note that it is still necessary to specify an INPUT object, because it is needed co construct the PCE basis.

2.5.2 Truncation options

To determine the truncation sets $\mathcal{A}_1, \ldots, \mathcal{A}_4$ in (1.13) and (1.14), degree and q-norm adaptivity are used as explained in Section 2. By default, the following ranges are used for the degree:

```
MetaOpts.Lambda(1).Degree = 0:5;
MetaOpts.Lambda(2).Degree = 0:3;
MetaOpts.Lambda(3).Degree = 0:1;
MetaOpts.Lambda(4).Degree = 0:1;
```

The following values are used for the q-norm:

```
MetaOpts.Lambda(1).TruncOptions.qNorm = 1;
MetaOpts.Lambda(2).TruncOptions.qNorm = 1;
MetaOpts.Lambda(3).TruncOptions.qNorm = 0.75;
MetaOpts.Lambda(4).TruncOptions.qNorm = 0.75;
```

It is also possible to specify range arrays with values between 0 and 1 for the q-norm.
A sparse PCE is computed for every combination of degree and q-norm (depending on the chosen regression method either on all λ_i's, or only on λ_1 and λ_2, (Zhu et al., 2022)), and the PCE with the smallest leave-one-out (LOO) error is chosen as the final metamodel.
For both degree and q-norm adaptivity, UQLAB implements a so-called early stop criterion: if the LOO error increases twice in a row for subsequent values of degree and q-norm, the adaptivity iteration is stopped. This behavior can be turned off by setting the following flags to `false`:

```
MetaOpts.DegreeEarlyStop = false; % default: true
MetaOpts.qNormEarlyStop = false; % default: true
```

2.5.3 Use of a validation set

A validation set can be provided as follows:

```
MetaOpts.ValidationSet.X = Xval;
MetaOpts.ValidationSet.Y = Yval;
```

UQLAB calculates various validation error metrics based on this data set, which are stored in `myGLaM.Error.Val`.

2.5.4 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 computes an independent GLaM for each output component on the shared experimental design. Therefore, the surrogate model can only be used to estimate

the marginal response distribution for each output component, and it should not be employed to capture their joint distribution. No additional configuration is needed to enable this behaviour.

A GLaM with multi-component outputs can be found in the UQLAB examples in:

`Examples/GLaM/uq_Example_GLaM_04_MultiOutputs.m`

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

```
>> myGLaM.GLaM

ans =
  1x8 struct array with fields:

    Lambda
    OutputId
    Basis
    Coefficients
    Transform
```

Its fields are explained in Table 11.

Each element of the `myGLaM` structure is functionally identical to its scalar counterpart in Section 2.4. Similarly, the `myGLaM.Error` structure becomes a multi-element structure:

```
>> myGLaM.Error

ans =
  1x2 struct array with fields:

    NLogLikelihood
    AIC
    AICc
    BIC
    KIC
    TIC
```

2.5.5 Method-specific options

2.5.5.1 `'RepJoint'`

The user can choose the inference method for the lambda distribution by specifying

```
MetaOpts.RepJoint.RepMethod = 'MLE';
```

The default is maximum likelihood estimation (`'MLE'`). For the various other available options, see Table 4. See Chalabi (2012) for a review of the different methods.

The regression method for each λ_i can be any of the available solvers for sparse PCE implemented in UQLAB, see UQLAB User Manual – Polynomial Chaos Expansions. By default, the method is `'LARS'`:

```
MetaOpts.RepJoint.RegMethod = {'LARS', 'LARS', 'LARS', 'LARS'};
```

2.5.5.2 'FullReg'

In the FGLS algorithm, the PCEs approximating mean and variance function are fit iteratively. The sparse regression solver can be changed, see UQLAB User Manual – Polynomial Chaos Expansions for a list of available solvers (default: 'LARS' as proposed by Zhu et al. (2022)). Furthermore, the user can determine whether the PCE basis of these two quantities should be re-selected in every FGLS iteration (true) or only once in the beginning (false):

```
MetaOpts.FullReg.Mean.Method = 'LARS';
MetaOpts.FullReg.Mean.updateBasis = false; % default: false
MetaOpts.FullReg.Var.Method = 'LARS';
MetaOpts.FullReg.Var.updateBasis = true; % default: true
```

Finally, the model selection criterion can be chosen from 'BIC' (default), 'AIC', corrected AIC 'AICc', and negative log-likelihood NLogLikelihood. See Ye et al. (2008) for a review of these criteria.

```
MetaOpts.FullReg.SelectCrit = 'BIC';
```

2.5.6 Using GLaM as a model (predictor)

Every GLaM metamodel computed by UQLAB is a MODEL object. It can be evaluated on points from the input domain just like any stochastic simulator implemented in UQLAB.
For example, we obtain an evaluation for a new sample point X by

```
X = uq_getSample(1);
Y_GLaM = uq_evalModel(myGLaM, X);
```

2.5.7 Custom GLaM

It is possible to manually define a GLaM by specifying the required PCE properties (*i.e.* basis and coefficients) for all four lambdas.
This can be achieved as follows:

```
CustomOpts.Type = 'Metamodel';
CustomOpts.MetaType = 'GLaM';
CustomOpts.Method = 'Custom';
CustomOpts.Input = myInput;
```

Assuming that the 1×4 struct array myCustomBasis has the fields .Indices, .PolyTypes, and .PolyTypesParams with data in the correct format (see UQLAB User Manual – Polynomial Chaos Expansions for more information), and another 1×4 cell array LambdaCoeffs contains the corresponding coefficient arrays of matching size, the lambda functions are specified as follows:

```
for ii=1:4
  CustomOpts.Lambda(ii).Basis = myCustomBasis(ii);
  CustomOpts.Lambda(ii).Coefficients = LambdaCoeffs{ii};
end
```

Finally, the custom GLam is created.

```
customGLaM = uq_createModel(CustomOpts);
```

No calculation is taking place.

Chapter 3

Reference List

How to read the reference list

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

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

which corresponds to the following syntax:

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

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

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

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

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

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

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

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

This is illustrated as follows:

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

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

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

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

3.1 Create a GLaM metamodel

Syntax

```
myGLaM = uq_createModel(MetaOpts)
```

Input

The struct variable `MetaOpts` contains the following fields:

Table 1: `MetaOpts`

●	`.Type`	`'Metamodel'`	Select the metamodelling tool
●	`.MetaType`	`'GLaM'`	Select the generalized lambda model
☐	`.Input`	INPUT object	Probabilistic input model
☐	`.Name`	String	Unique identifier for the metamodel
☐	`.Display`	String default: `'standard'`	Level of information displayed by the methods.
		`'quiet'`	Minimum display level, displays nothing or very few information.
		`'standard'`	Normal display level, shows the most important information.
		`'verbose'`	Maximum display level, shows all the information on runtime, like updates on iterations, etc.
☐	`.FullModel`	MODEL object	UQLab model to be surrogated
☐	`.Lambda`	1×4 struct array, see Table 2	Basis specification for the PCE modelling of the lambda functions
☐	`.Method`	String default: `'FullReg'` if number of replications ≤ 50, `'RepJoint'` otherwise	Fitting method
		`'RepJoint'`	Replication-based method
		`'FullReg'`	Maximum likelihood estimation with full basis selection for λ_3 and λ_4
⊞	`.RepJoint`	Table 4	Options for the method `RepJoint`
⊞	`.FullReg`	Table 5	Options for the method `FullReg`
●	`.ExpDesign`	Table 8	Experimental design-specific options
☐	`.ValidationSet`	Table 9	Validation set components

☐	`.DegreeEarlyStop`	Logical default: true	Whether or not to use the option "DegreeEarlyStop" (see UQLAB User Manual – Polynomial Chaos Expansions)
☐	`.qNormEarlyStop`	Logical default: true	Whether or not to use the option "qNormEarlyStop" (see UQLAB User Manual – Polynomial Chaos Expansions)

Note: If `.FullModel` or `.Input` are not specified, the currently active model or input is used. By default, the *last created* model or surrogate model is the currently active model.

3.1.1 Lambda options

The properties of the four PCEs for $\lambda_1, \ldots, \lambda_4$ are defined in the 1×4 struct array `.Lambda`.

Table 2: `MetaOpts.Lambda`

●	`.Degree`	Defaults: For λ_1: `0:5` For λ_2: `0:3` For λ_3: `0:1` For λ_4: `0:1`	Degree(s) for the four PCEs modeling the parameters
		Integer scalar	Maximum polynomial degree
		Integer array	Set of polynomial degrees for degree-adaptive polynomial chaos (see UQLAB User Manual – Polynomial Chaos Expansions)
☐	`.TruncOptions`	Table 3	Basis truncation (see UQLAB User Manual – Polynomial Chaos Expansions)

Table 3: `Lambda.TruncOptions`

☐	`.qNorm`	Double Defaults: 1 for λ_1, λ_2 0.75 for λ_3, λ_4	Hyperbolic truncation
		Double array	Set of hyperbolic norms for q-norm-adaptive polynomial chaos (see UQLAB User Manual – Polynomial Chaos Expansions).
☐	`.MaxInteraction`	Integer default: M	Maximum rank truncation
☐	`.Custom`	$P \times M$ integer array	Manual specification of the multi-index set $\mathcal{A}$

3.1.2 Method-specific options

The options in the following table are specific for the `'RepJoint'` method.

Table 4: `MetaOpts.RepJoint`			
☐	`.RepMethod`	String default: `'MLE'`	Inference method for determining the best parameters $\lambda_1, \ldots, \lambda_4$ from the replications for each point in the experimental design
		`'MM'`	Method of moments
		`'MLE'`	Maximum likelihood estimation
☐	`.RegMethod`	1×4 cell array of strings default: all `'LARS'`	Regression method for fitting PCEs to $\lambda_1, \ldots, \lambda_4$ (see UQLAB User Manual – Polynomial Chaos Expansions for an overview of available methods)

The options in the following table are specific for the `'FullReg'` method.

Table 5: `MetaOpts.FullReg`			
☐	`.Mean`	Struct, see Table 6	Options for the fitting of the mean function
☐	`.Var`	Struct, see Table 7	Options for the fitting of the variance function
☐	`.SelectCrit`	String default: `'BIC'`	Selection criterion for the GLaM
		`'NLogLikelihood'`	Negative log-likelihood
		`'AIC'`	Akaike information criterion
		`'AICc'`	corrected AIC
		`'BIC'`	Bayesian information criterion

Table 6: `FullReg.Mean`			
☐	`.Method`	String default: `'LARS'`	Sparse regression solver
☐	`.updateBasis`	Logical default: false	Whether the basis should be updated in every FGLS iteration (`true`) or not

Table 7: `FullReg.Var`			
☐	`.Method`	String default: `'LARS'`	Sparse regression solver
☐	`.updateBasis`	Logical default: true	Whether the basis should be updated in every FGLS iteration (`true`) or not

3.1.3 Experimental design

Table 8: `MetaOpts.ExpDesign`			
⊕	`.Sampling`	String default: `'LHS'`	Sampling type for points in the input space
		`'MC'`	Monte Carlo sampling
		`'LHS'`	Latin Hypercube sampling
☐	`.NSamples`	Integer	The number of samples to draw. It is required when `.Sampling` is specified.
☐	`.Replications`	Integer	The number of replications. It is required if `'RepJoint'` is chosen as the fitting method.
⊕	`.X`	$N \times M$ Double	User-defined experimental design X. If specified, `.Sampling` is ignored.
⊕	`.Y`	$N \times N_{\text{out}} \times R$ Double	User-defined model response Y with R replications. If specified, `.Sampling` is ignored.

3.1.4 Validation Set

Table 9: `MetaOpts.ValidationSet`			
●	`.X`	$N \times M$ double	User-specified validation set $\boldsymbol{\mathcal{X}}_{\text{Val}}$
●	`.Y`	$N \times N_{\text{out}} \times R$ double	User-specified validation set response $\mathcal{Y}_{\text{Val}}$ with replications

3.2 Accessing the results

Syntax

```
myGLaM = uq_createModel(MetaOpts);
```

Output

Regardless on the configuration options given at creation time in the `MetaOpts` structure, all GLaM metamodels share the same output structure, given in Table 10.

Table 10: `myGLaM`		
`.Name`	String	Unique name of the GLaM metamodel
`.Options`	Table 1	Copy of the `MetaOpts` structure used to create the metamodel

`.GLaM`	$1 \times (4N_{\text{out}})$ struct array with fields detailed in Table 11	Information about the final GLaM model
`.ExpDesign`	Table 13	Experimental design used for calculating the coefficients
`.Error`	$1 \times N_{\text{out}}$ struct array with fields detailed in Table 12	Error estimates of the metamodels's accuracy
`.Internal`	Table 14	Internal state of the MODEL object (useful for debug/diagnostics)

The field `myGLaM.GLaM` contains a $1 \times (4N_{\text{out}})$ struct array defining the PCEs modeling the four parameters of the GLaM, for each of the N_{out} outputs of the computational model.

Table 11: `myGLaM.GLaM`		
`.Lambda`	Integer	Index i of the GLaM parameter λ_i defined by this struct
`.OutputId`	Integer	Index of the model output
`.Basis`	Struct array	Basis information for the PCE of λ_i, see UQLAB User Manual – Polynomial Chaos Expansions
`.Coefficients`	Double array	Coefficient vector of the PCE of λ_i
`.Transform`	Struct	Definition of the transform used for modeling λ_i

Table 12: `myGLaM.Error`		
`.NLogLikelihood`	Double	Negative log-likelihood
`.AIC`	Double	Akaike information criterion
`.AICc`	Double	corrected AIC
`.BIC`	Double	Bayesian information criterion
`.KIC`	Double	Kashyap information criterion
`.TIC`	Double	Takeuchi's information criterion
`.Val`	Struct	Error metrics between the GLaM surrogate and the provided validation set, computed by `uq_evalStochSimMetrics` (see Section 3.3.2)

Table 13: `myGLaM.ExpDesign`		
`.NSamples`	Integer	The number of samples
`.Sampling`	String	The sampling method

.ED_Input	INPUT object	The input module that was used to generate the experimental design
.Replications	Integer	Number of replications
.isTrajectory	Logical	Whether or not the experimental design consists of trajectories
.X	$N \times M$ double	The experimental design values
.U	$N \times M$ double	The experimental design values in the reduced space
.Y	$N \times N_{out} \times R$ double	The output Y that corresponds to the input X, with R replications

The following table contains a selection of fields from `myGLaM.Internal` that might be of interest for the user.

Table 14: `myGLaM.Internal`

.Input	INPUT object	The input module that was used to generate the experimental design
.FullModel	MODEL object	The model object
.Method	String	Method used for fitting the coefficients
.DegreeEarlyStop	Logical	Whether or not the option "DegreeEarlyStop" was used (see UQLAB User Manual – Polynomial Chaos Expansions)
.qNormEarlyStop	Logical	Whether or not the option "qNormEarlyStop" was used (see UQLAB User Manual – Polynomial Chaos Expansions)
.Lambda	1×4 struct array	Further options for PCE fitting of parameters
.Basis	Struct	Information about the polynomial families used for the PCEs
.(Method)	Struct	Options specific to the chosen fitting method
.ExpDesign	Struct	More information about the experimental design

3.3 Additional functions

3.3.1 uq_display

Syntax

```
uq_display(GLaModel)
uq_display(GLaModel, OUTARRAY)
uq_display(GLaModel, 'lambda')
```

Description

Note: This display function only works for GLaM models with 1- and 2-dimensional inputs.

`uq_display(GLaModel)` plots the median and the 2.5%-97.5% quantile (for 1-dimensional input) or the mean and variance (for 2-dimensional input) of a GLaM predictor specified by `GLaModel`. If there is more than one output, only the first output component is plotted.

`uq_display(GLaModel, OUTARRAY)` creates the plots of a GLaM predictor with multiple outputs for the selected output components given in OUTARRAY.

`uq_display(GLaModel, 'lambda')` plots the parameter functions $\lambda_i(\boldsymbol{x})$.

3.3.2 uq_evalStochSimMetrics

Syntax

```
uq_evalStochSimMetrics(GLaModel, X, Y)
uq_evalStochSimMetrics(GLaModel, X, Y, Name, Value)
Metrics = uq_evalStochSimMetrics(...)
```

Description

`uq_evalStochSimMetrics(GLaModel, X, Y)` computes error metrics between a generalized lambda model and a data set (X, Y). Here, X is an experimental design sampled from the input space, and Y contains evaluations of a stochastic model with replications.

`uq_evalStochSimMetrics(GLaModel, X, Y, Name, Value)` allows for fine-tuning the metrics computation by specifying `Name`, and `Value` pairs of options. The available options are summarized in Table 15.

`Metrics = uq_evalStochSimMetrics(...)` returns a struct with fields corresponding to different error metrics. These include the negative log-likelihood (1.16), the normalized Wasserstein distance, and the Wasserstein distances for every sample point. For the definitions of Wasserstein distances, the reader is referred to Zhu and Sudret (2021a).

Table 15: Available options of `uq_evalStochSimMetrics`

Name	Value (default)	Description
`'WSDorder'`	Double (p) default: 2	Order of the Wasserstein distance
`'WSDcalcMethod'`	String default: `'ot'`	Method to compute the Wasserstein distance
	`'quantile'`	Calculation based on the empirical quantile

	'mixed'	Calculation based on mean, standard deviation and quantile correlations (Schefzik et al., 2021)
	'ot'	Exact semi-discrete Wasserstein distance based on optimal transport (Solomon, 2018)
'TruncLambdai'	Double array ([l,u])	The lambda function $\lambda_i(\boldsymbol{x})$ is truncated to stay within the interval $[l, u]$. This can be necessary for the Wasserstein distance to be finite. Replace 'i' by the respective number, e.g. 'TruncLambda4'.
'TruncLambda'	Double array [l_1,u_1;l_2,u_2; l_3,u_3;l_4,u_4]	Set the bounds for $\lambda_1, \ldots, \lambda_4$ all together. The i-th row corresponds to the bounds for λ_i.

Usage

```
% Create a validation set:
Nval = 1e3;
Xval = uq_getSample(myInput, Nval, 'LHS');
% use 1e4 replications to represent the response distribution
Yval = uq_evalModel(myModel, Xval, 1e4);

% Evaluate the normalized Wasserstein distance on the validation set:
valMetrics = uq_evalStochSimMetrics(myGLaM, Xval, Yval);
fprintf('Normalized Wasserstein distance: %1.3e. \n', ...
    valMetrics.NormalizedWSD);
```

3.3.3 uq_GLaM_evalLambda

Syntax

```
lambda = uq_GLaM_evalLambda(GLaModel, X)
lambda = uq_GLaM_evalLambda(GLaModel, X, Name, Value)
```

Description

`lambda = uq_GLaM_evalLambda(GLaModel, X)` evaluates the parameters $\lambda_1, \ldots, \lambda_4$ for a given generalized lambda model `GLaModel` at the points specified in `X`. `lambda` is an array of size $N \times N_{\text{out}} \times 4$, where N is the number of rows of `X`.

`lambda = uq_GLaM_evalLambda(GLaModel, X, Name, Value)` allows for fine-tuning the evaluation by specifying `Name-Value` pairs of options, see Table 16.

Table 16: Available options of `uq_GLaM_evalLambda`

Name	Value (default)	Description

`'TruncLambdai'`	Double array (`[l,u]`)	The lambda function $\lambda_i(\boldsymbol{x})$ is truncated to stay within the interval $[l, u]$. This can be necessary for the Wasserstein distance to be finite. Replace `'i'` by the respective number, e.g. `'TruncLambda4'`.
`'TruncLambda'`	Double array `[l_1,u_1;l_2,u_2; l_3,u_3;l_4,u_4]`	Set the bounds for $\lambda_1, \ldots, \lambda_4$ all together. The i-th row corresponds to the bounds for λ_i.

Usage

```
% Define the points to visualize the associated PDFs
Xplot = [0.07, 0.13; 0.04, 0.12; 0.05, 0.3; 0.02, 0.33];

% Generate samples from the GLaM
R = 1e5; % number of replications
YplotGLaM = uq_evalModel(myGLaM, Xplot, R);

% Evaluate the lambda functions on the plotting points
lambda = uq_GLaM_evalLambda(myGLaM, Xplot);
```

3.3.4 uq_GLD_DistributionFunc

Syntax

```
[funcVal,xVal] = uq_GLD_DistributionFunc(lambda, func, xVal)
```

Description

Evaluates a lambda distribution with parameters specified in `lambda`. The argument `func` can be `'pdf'`, `'cdf'` or `'quantile'`. If no third argument is provided, the array

```
u=(0:0.0001:1)'
```

in quantile space is used for evaluation.

Usage

```
[yplot, xplot] = uq_GLD_DistributionFunc(lambda, 'pdf');
```

References

Chalabi, Y. (2012). *New directions in statistical distributions, parametric modeling and portfolio selection*. PhD thesis, ETH Zürich, Switzerland. 4, 15

Freimer, M., Kollia, G., Mudholkar, G. S., and Lin, C. T. (1988). A study of the generalized Tukey lambda family. *Comm. Stat. Theor. Meth.*, 17:3547–3567. 2

Schefzik, R., Flesch, J., and Goncalves, A. (2021). Fast identification of differential distributions in single-cell RNA-sequencing data with waddR. *Bioinformatics*, 37(19):3204–3211. 28

Solomon, J. (2018). Optimal transport on discrete domains. *AMS Short Course on Discrete Differential Geometry*. 28

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. 3

Wooldridge, J. M. (2013). *Introductory Econometrics: A Modern Approach*. 3 edition. 7

Xiu, D. and Karniadakis, G. E. (2002). The Wiener-Askey polynomial chaos for stochastic differential equations. *SIAM Journal of Scientific Computing*, 24(2):619–644. 3

Ye, M., Meyer, P. D., and Neuman, S. P. (2008). On model selection criteria in multimodel analysis. *Water Resources Research*, 44(3). 13, 16

Zhu, X., Broccardo, M., and Sudret, B. (2022). Use of generalized lambda models for seismic fragility analysis. In *Proc. 8th International Symposium on Reliability Engineering and Risk Management (ISRERM), Hannover (Germany), September 4-7*. 1, 7, 14, 16

Zhu, X. and Sudret, B. (2020). Replication-based emulation of the response distribution of stochastic simulators using generalized lambda distributions. *International Journal for Uncertainty Quantification*, 10(3):249–275. 1, 5, 6

Zhu, X. and Sudret, B. (2021a). Emulation of stochastic simulators using generalized lambda models. *SIAM/ASA Journal on Uncertainty Quantification*, 9(4):1345–1380. 1, 3, 5, 6, 7, 27

Zhu, X. and Sudret, B. (2021b). Global sensitivity analysis for stochastic simulators based on generalized lambda surrogate models. *Reliability Engineering & System Safety*, 214:107815. 1, 5