Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Uncertainty Quantification in Engineering

Mini-Project

Start date: April 3, 2025
Hand-in date: May 23, 2025

Introduction

In this assignment, you will develop your own Polynomial Chaos Expansion tool and apply it to a case study of your choice. The exercise is divided into several sections, each dealing with a different aspect of the problem.

You are required to program the problem solutions from scratch using Matlab – do not use any already pre-existing code for uncertainty quantification. A set of helper codes to handle the most technically challenging sections is provided as an aid.

The project is done in groups of 3-4 students and supervised by a teaching assistant. It lasts in total 7 weeks.

Deliverables

By May 23, submit the following documents by email to your supervisor (with nluethen@ethz.ch in CC):

1. The **project report** explaining the computational model, the input, the PCE computation, and your numerical experiments as well as their results. It should also include a conclusion, in which you explain the insights you got from your numerical experiments.
    - The report is submitted electronically as PDF and must have **at most 10 pages** (not counting the title page and the declaration of originality).
    - The report should clearly indicate the contribution of each student in the group and must include a declaration of originality.

- The report needs to follow ETH's citation etiquette. Furthermore, it is not allowed to incorporate text generated by artificial intelligence software (e.g., ChatGPT) without any changes. If such software was used to aid writing the report, it must clearly be stated which parts of the report are concerned and in which way the software was used.

2. The **Matlab code** you wrote for the project. (The beauty of your code does not matter for your grade.)

For information on how the project will be graded, see Section 3.

Important remarks

- This is a **mini-project**, not the usual exercise-based homework. It is graded and you are expected to interact with your assigned supervisor at least twice: once in the kick-off meeting at the start of the project, and at least once in the middle of your project to demonstrate your progress and clarify open questions.
- Meetings with the supervisor should always include all group members. The meetings can take place in person or online through Zoom.
- It is a mini-project, not a semester thesis. You should spend a reasonable amount of time on it but not several weeks full-time. If you get stuck or have other problems, contact your supervisor early on.
- The whole group gets the same grade. In case there is a problem with a team member, contact your supervisor.
- The project consists of the following components:
  - Implementing the model, the input, and the isoprobabilistic transform
  - Implementing the PCE
  - Performing numerical experiments
  - Writing the report
  - For groups with 4 members: comparison with UQLab

  For many groups in the past years, it worked well to have one team member responsible for each of the parts. But you can divide the work among yourselves however you want. We strongly encourage you to discuss the concepts and your solutions with your team members throughout the project duration, so that you benefit as much as possible from this exercise.
- This document contains the minimum requirements for the mini-project. You are free to do more, but make sure you have done at least the minimum.

1 Overview

The focus of this mini-project is to develop a polynomial chaos expansion tool for metamodelling a model of your choice. For the sake of simplicity, some technical options are restricted to "simple" methods, but you are free to extend your tool once the minimum requirements for the mini-project are met.

1.1 Ingredients needed in PCE

The following bullets represent the various bricks that need to be coded in order to complete a polynomial chaos expansion tool.

- A computational model that can be surrogated
- A probabilistic input model
- A truncated orthonormal polynomial basis
- Sampling an experimental design
- Computing the corresponding polynomial coefficients
- The evaluation of the computed expansion for new values of the input parameters

2 Detailed project description

Reminder: the truncated polynomial chaos expansion of a model $Y = \mathcal{M}(\boldsymbol{X})$ is given by:

$$Y = \mathcal{M}(\boldsymbol{X}) \approx \mathcal{M}^{\mathsf{PCE}}(\boldsymbol{X}) = \sum_{\boldsymbol{\alpha} \in \mathcal{A}} c_{\boldsymbol{\alpha}} \, \Psi_{\boldsymbol{\alpha}}(\boldsymbol{X}) \tag{1}$$

where $c_{\boldsymbol{\alpha}}$ are real numbers and $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X})$ are multivariate polynomials orthonormal with respect to the joint PDF of $\boldsymbol{X}$.

2.1 Choice and implementation of a model

Each group is allowed to come up with their own computational model to surrogate, as long as the following conditions are met:

- The number of input parameters M is between 3 and 5.

- The model is continuous and smooth for all the plausible values of the inputs. The model should not be polynomial. The model should preferably be an analytical formula.
- The model is a *scalar-valued* function (i.e., for each sample of the M input variables, only one output is produced).
- The model is *vectorized*. This means that, given an $n \times M$ matrix X of n realizations of an input vector $\boldsymbol{x} \sim \boldsymbol{X}$ (each row of X containing one realization), the model will return an $n \times 1$ vector Y of model responses $y = \mathcal{M}(\boldsymbol{x}) \in \mathbb{R}$, with the following syntax: `Y = myCustomModel(X);`

If you have no idea what model to use: You can use, e.g., the dyke problem from the homework exercises of week 4.

Note: For easier validation of your elementary code bricks, you can test your implementation with a simple polynomial model.

2.2 The probabilistic input model

First of all, choose a suitable probabilistic model of the input $\boldsymbol{X}$ to your computational model: To limit the complexity of this project, assume that the input variables are independent, and that each marginal follows either

- a uniform distribution, $X_i \sim \mathcal{U}([a_i,\ b_i])$; or
- a Gaussian distribution, $X_j \sim \mathcal{N}(\mu_j, \sigma_j)$.

Choose the parameters of the marginal distributions to be reasonable for the modelled quantity (your own judgment!).

Justify all your choices in the report.

Hint: If you want to keep it simple, use uniform distributions for all input variables. You can always modify your code later to allow for Gaussian distributions.

2.3 Sampling the experimental design

The method that shall be used for the calculation of the polynomial coefficients is **least-squares regression**. The experimental design can therefore be created independently from the remaining PCE components. In order to create an experimental design $\mathcal{X}_{\mathsf{ED}} = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)}\}$, you need to implement the following steps:

- **Choice of a sampling strategy:** standard Monte Carlo sampling or Latin Hypercube Sampling. MATLAB has built-in functions to create samples in the unit hypercube, see Section A.1 for a reference.
- **Sampling of the input variables according to their distributions:** the functions mentioned in the previous point produce samples from the unit hypercube: $\boldsymbol{u} \sim \mathcal{U}([0,1]^M)$. Apply the correct isoprobabilistic transform to obtain a sample from the input vector $\boldsymbol{X}$ defined in Section 2.2.
- **Evaluation of the full model on the experimental design:** evaluate the model from Section 2.1 on the generated experimental design. Make sure the resulting responses are stored in matrix format (response vector $\boldsymbol{y}_{\text{ED}} = (y_1, y_2, ..., y_n)^T$ with $y_i = \mathcal{M}(\boldsymbol{x}^{(i)})$).

We recommend to use at least $n = 100$ samples as a starting point.

2.4 The polynomial basis

Next, you have to implement the multivariate polynomial basis. This part is most likely the most complex to develop, due to the necessity of coding tensor products effectively (book-keeping problem). It is divided into several subsections, each one dealing with one aspect of the polynomial basis generation. Before proceeding, you must select a maximum polynomial degree p for the PCE calculations. We recommended to start with low-degree polynomials ($1 \leq p \leq 4$) and gradually increase, to keep complexity manageable.

2.4.1 Univariate Legendre or Hermite polynomials

Because the input variables follow a uniform or Gaussian distributions, the univariate polynomials in the PCE are Legendre and/or Hermite polynomials.

We provide you with the implementation of these univariate orthonormal polynomials:

- Legendre polynomials defined on $[-1, 1]$ in the provided file `eval_legendre.m`
- Hermite polynomials orthonormal w.r.t. $\mathcal{N}(0, 1)$ in the provided file `eval_hermite.m`

See Section A.2 for the usage of these functions.

2.4.2 Evaluate the ED on univariate polynomials

In order to perform least-square regression, the univariate orthonormal polynomials up to degree p need to be evaluated on the experimental design $\mathcal{X}_{\text{ED}}$.

Classical Legendre and Hermite polynomials, however, are orthonormal only with respect to the densities with standard parameters, i.e., $[-1, 1]$ for the uniform distribution and $[0, 1]$ for the Gaussian distribution:

$$\xi \sim \mathcal{U}([-1, 1]) \text{ or } \xi \sim \mathcal{N}(0, 1) \tag{2}$$

Therefore, the experimental design calculated in Section 2.3 must first be transformed with a suitable isoprobabilistic transform:

$$\boldsymbol{\xi}^{(i)} = \boldsymbol{\mathcal{T}}(\boldsymbol{x}^{(i)}) \tag{3}$$

before the Legendre or Hermite polynomials can be correctly evaluated on it.

2.4.3 Polynomial degree and basis truncation

Given a set of evaluations of univariate polynomials $\psi_k(x_i)$, with $0 < k \leq p$ and $1 < i \leq M$, it is possible to create a multivariate polynomial basis by tensor product of the univariate ones according to:

$$\Psi_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \prod_{i=1}^{M} \psi_{\alpha_i}(x_i) \tag{4}$$

where $\boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_M\}$. Employing total degree truncation, $|\boldsymbol{\alpha}| \overset{\text{def}}{=} \alpha_1 + ... + \alpha_M \leq p$.

You have generated the evaluations of the univariate polynomials $\psi_k(x_i)$ in the previous step, so what remains is to create all the possible multi-indices $\boldsymbol{\alpha}$ that satisfy the total degree truncation requirement $|\boldsymbol{\alpha}| \leq p$. The cardinality of this set of indices will be denoted by P.

Efficiently performing this operation is computationally and algorithmically intensive, and not particularly instructive in this context, hence you should make use of the provided function `create_alphas(M,p)` (see also Section A.3) to generate the set of $\boldsymbol{\alpha} \in \mathcal{A}$ needed to create the regression matrix $\boldsymbol{\Psi}$. This matrix has entries $\boldsymbol{\Psi}_{ij} = \Psi_j\left(\boldsymbol{x}^{(i)}\right)$, where $1 \leq j \leq P$ is the index over the P basis elements generated by the `create_alphas(M,p)` function, while $1 \leq i \leq n$ is the index over the elements of the experimental design. The full matrix is given by

$$\boldsymbol{\Psi} = \begin{pmatrix} \Psi_1\left(\boldsymbol{x}^{(1)}\right) & \cdots & \Psi_P\left(\boldsymbol{x}^{(1)}\right) \\ \vdots & \ddots & \vdots \\ \Psi_1\left(\boldsymbol{x}^{(n)}\right) & \cdots & \Psi_P\left(\boldsymbol{x}^{(n)}\right) \end{pmatrix}. \tag{5}$$

2.5 Calculation of the polynomial coefficients with Ordinary Least-Squares (OLS)

Now that the full set of basis elements $\{\Psi_{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha} \in \mathcal{A}}$ has been evaluated on the experimental design $\mathcal{X}_{\text{ED}}$, which yielded the regression matrix $\boldsymbol{\Psi}$, the calculation of the coefficients via least-square minimization is reduced to an algebraic problem that is easily solved in MATLAB.

Indeed, the vector $\boldsymbol{c}$ containing the PCE coefficients can be simply obtained from the matrix $\boldsymbol{\Psi}$ and the response vector $\boldsymbol{y}_{\text{ED}}$ as:

$$\boldsymbol{c} = \left(\boldsymbol{\Psi}^{\mathsf{T}}\boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Psi}^{\mathsf{T}}\boldsymbol{y}_{\text{ED}} \tag{6}$$

Use Matlab's built-in least-squares solver `mldivide`, also known as backslash operator, to compute the solution $\boldsymbol{c}$ to Eq. (6). Read the documentation to learn how to use it.

2.6 Numerical experiments

Now that you have implemented the input, the model and the PCE, it is time to validate that everything works correctly, and to analyze the performance of the PCE for your toy model.

Perform the following numerical experiments. You are free to come up with other interesting experiments.

Always think about what insights you are gaining through your experiments. In the report, show relevant plots, report your observations, and discuss the results.

2.6.1 Y-Y plots

Your PCE $\mathcal{M}^{\text{PCE}}$ was computed based on the experimental design $\mathcal{X}_{\text{ED}}$ of size n. As a first experiment, do the following:

1. **Y-Y plots.** Create Y-Y plots of PCE evaluations against model evaluations for two different values of the PCE degree p. Evaluate both models ($\mathcal{M}$ and $\mathcal{M}^{\text{PCE}}$) on the experimental design $\mathcal{X}_{\text{ED}}$ of size n.

2.6.2 Global error (see Lecture 8)

As you know from Lecture 8, the global error can be computed using the following three methods:

- The relative *empirical error*, evaluated on the experimental design $\mathcal{X}_{\text{ED}}$ that was used to compute the PCE
- The relative *leave-one-out error* (LOO), likewise computed using the experimental design $\mathcal{X}_{\text{ED}}$

- The *validation error* (relative mean-squared error) evaluated on a *validation set* $\mathcal{X}_V$.

  For this, create a large *validation set* $\mathcal{X}_V = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n_V)}\}$ of size $n_V = 10^5$ or 10^6 and evaluate the computational model $\mathcal{M}$ as well as the metamodel $\mathcal{M}^{\text{PCE}}$ on it. You only need to create this set once, and then you can reuse it.

  Note: Recall again that Legendre and Hermite polynomials are not orthonormal w.r.t. the input random vector $\boldsymbol{X}$, but w.r.t. the transformed vector $\boldsymbol{\Xi} = \mathcal{T}(\boldsymbol{X})$ with $\Xi_i \sim \mathcal{U}([-1, 1])$ or $\Xi_i \sim \mathcal{N}(0, 1)$. Hence, to evaluate the PCE on new samples $\boldsymbol{x} \sim \boldsymbol{X}$, it is necessary to pay attention to operating with the correct variable space (isoprobabilistic transforms).

Investigate now the behavior of the global error of your PCE for different values of experimental design size n and total degree p. Perform the following two experiments:

2. **Fixed n, increasing p.** Keep $n = 5 - 10M$ fixed and increase the total degree p, starting from $p = 1$. Increase p until $P \geq n$ (see Section 2.4.3 for the definition of p and P). For each p, compute the associated PCE and its relative empirical, LOO, and validation error. Plot the three errors against the increasing degree into one figure. Use log-scale for the y-axis.

3. **Fixed p, increasing n.** Keep p fixed to a reasonable, not too high value. Increase the experimental design size n, starting from $n = P$, up to $n = 4P$. You can use 5-10 increments, each time adding several tens of points. For each n, compute the associated PCE and its relative empirical, LOO, and validation error. Plot the three errors against the increasing experimental design size into one figure. Use log-scale for the y-axis.

In your report, describe and explain your observations in detail.

2.6.3 Relative error in mean and variance

Compute the mean and the variance of your model $\mathcal{M}$ either analytically, or by Monte Carlo simulation (using the large validation set generated previously).

Investigate now the convergence of the moments of your PCE for different values of experimental design size n and total degree p. Perform the following two experiments:

4. **Fixed n, increasing p.** Keep $n = 5 - 10M$ fixed and increase p, starting from $p = 1$. Increase p until $P \geq n$. For each p, compute the associated PCE and its mean and variance by postprocessing the coefficients. Plot the *relative errors* in mean and variance against the increasing degree into one figure. Use log-scale for the y-axis.

5. **Fixed p, increasing n.** Keep p fixed to a reasonable, not too high value. Increase the experimental design size n, starting from $n = P$, up to $n = 4P$. You can use 5-10

increments, each time adding several tens of points. For each n, compute the associated PCE and and its mean and variance by postprocessing the coefficients. Plot the *relative errors* in mean and variance against the increasing experimental design size into one figure. Use log-scale for the y-axis.

In your report, describe and explain your observations in detail.

2.7 For groups with 4 members: Comparison to UQLab

If your group has 4 members, you have one more task to do: compare your results to the results obtained with the Matlab-based uncertainty quantification software UQLab (https://www.uqlab.com/). An introduction to PCE with UQLab will be given in Tutorial 2. An extensive user manual for PCE with UQLab is available at https://www.uqlab.com/pce-user-manual.

Perform the following two experiments:

6. **Validation.** Use the same experimental design and the same total degree. Compute a least-squares PCE once with your own code, and once with UQLab. Make sure that the results coincide.
7. **Exploration.** Explore *one* of the more advanced features of PCE with UQLab which you find relevant to your problem. For example:
   - Choose a different, more realistic distribution for the input
   - Explore the effect of hyperbolic truncation
   - Compute the coefficients using a sparse solver such as LARS or OMP
   - Compute a surrogate using another surrogate modelling method, such as Kriging
   - Perform a sensitivity analysis

Describe and interpret the outcome of your experiments.

3 Grading information

Your mini project will be graded based on the following criteria:

- Results
  - Correct implementation of model, input, and sampling of the experimental design
  - Correct implementation of regression-based PCE
  - Correct implementation of validation error and moments
  - Execution of meaningful numerical experiments (Section 2.6) and creation of associated plots
  - Correctness of numerical results
- Report
  - Clear description of model and input, and justification for the choice of parameters
  - Clear but short description of PCE methodology
  - Demonstration of correct understanding of the core concepts (experimental design, isoprobabilistic transform, computation of the coefficients, validation error, moment computation, etc.)
  - Clear explanation of the numerical experiments (including all necessary details, e.g., about degree, experimental design and validation set sizes), description and interpretation of the outcomes
  - Conclusion: implications of the numerical results for the PCE methodology and the UQ problem
  - If applicable: topics discussed with supervisor must be addressed
  - Structure of report
  - Orthography and English language
- Interaction with the supervisor and clarification of questions and problems early on

A Appendix: Matlab function reference

A.1 Random samples: `rand` and `lhsdesign`

The basic function to generate random numbers in MATLAB is `rand`. It can be used to generate a single random number when called without arguments, or it can be used to generate an $n \times M$ matrix of random numbers uniformly distributed in $(0, 1)$ by typing:

```
A = rand(n,M);
```

However, in the statistics toolbox, MATLAB also offers the possibility to create a Latin Hypercube Sampling with the function `lhsdesign`. The syntax is similar to the previous one:

```
X = lhsdesign(n, M);
```

Note Both `rand` and `lhsdesign` are MATLAB functions, so you can see a more detailed description of their usage by typing `help rand` and `help lhsdesign` respectively.

A.2 Evaluate polynomials: `eval_legendre`

The function `eval_legendre(X, p)` evaluates the *normalized* univariate Legendre polynomial (orthonormal w.r.t. $\mathcal{U}([-1, 1])$) of order p on the coordinates given on a column vector X. For example, let us evaluate the normalized second order Legendre polynomial, which is given by (see lecture 6):

$$\tilde{P}_2(x) = \sqrt{5} \cdot \frac{(3x^2 - 1)}{2},$$

on the points $x_1 = -0.5$ and $x_2 = 0.3$. This can be done with the following code:

```
>> X = [-0.5; 0.3];
>> Y = eval_legendre(X, 2)

Y =
   -0.2795
   -0.8162
```

The function `eval_hermite` works analogously.

A.3 Generate a set of indices: `create_alphas`

The function `create_alphas(M, p)` returns a matrix with M columns, that contains in its rows all the possible combinations of $M-$dimensional vectors $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_M)$ such that $|\boldsymbol{\alpha}| = \alpha_1 + \cdots + \alpha_M \leq p$. Here is an example of its usage in dimension $M = 2$, for finding the indices that generate a polynomial basis with elements up to degree $p = 3$:

```
>> idx = create_alphas(2, 3)
```

```
idx =
 0     1
 1     0
 0     2
 2     0
 0     3
 3     0
 1     1
 1     2
 2     1
```